ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA E TELECOMUNICAZIONI PER LO SVILUPPO SOSTENIBILE

Generatore di entropia basato su dinamica caotica con interfaccia USB

TESI IN

Elettronica per l'Elaborazione Analogica del Segnale

RELATORE
Ing. Sergio Callegari

PRESENTATA DA
Mattia Fabbri

II Appello - III Sessione
Anno Accademico 2012/2013

Alla mia famiglia e ai miei amici.

«...we needn't accomplish great things, we only need to accomplish little things that make us feel better or not so bad.»
[Charles Bukowski]

Sinossi

È impossibile implementare sorgenti autenticamente casuali su hardware digitale. Quindi, storicamente, si è fatto ampio uso di generatori di numeri pseudo-casuali, evitando così i costi necessari per la progettazione di hardware analogico dedicato. Tuttavia, le sorgenti pseudo-casuali hanno proprietà (riproducibilità e periodicità) che si trasformano in vulnerabilità, nel caso in cui vengano adottate in sistemi di sicurezza informatica e all'interno di algoritmi crittografici. Con l'esperienza acquisita a valle di alcuni incidenti causati da un uso scorretto dei generatori pseudo-casuali, oggi la richiesta di generatori di numeri autenticamente casuali è ai suoi massimi storici. Alcuni importanti attori dell'Information and Communication Technology hanno sviluppato proprie soluzioni dedicate, ma queste sono disponibili solo sui sistemi moderni e di fascia elevata. È quindi di grande attualità rendere fruibili generatori autenticamente casuali per sistemi già esistenti o a basso costo. Per garantire sicurezza e al tempo stesso contenere i costi di progetto è opportuno pensare ad architetture che consentano di riusare parti analogiche già disponibili. Particolarmente interessanti risultano alcune architetture che, grazie all'utilizzo di dinamiche caotiche, consentono di basare buona parte della catena analogica di elaborazione su convertitori analogico/digitali. Infatti, tali blocchi sono ampiamente fruibili in forma integrata, anche come unità funzionali di sistemi più complessi come, per esempio, architetture programmabili e microcontrollori.

In questo lavoro, si propone un'implementazione a basso costo ed elevata flessibilità di un architettura basata su un convertitore analogico/digitale, inizialmente concepita all'Università di Bologna. La riduzione di costo viene ottenuta sfruttando il convertitore già presente all'interno di un microcontrollore. L'elevata flessibilità deriva dal fatto che il microcontrollore prescelto mette a disposizione una varietà di interfacce di comunicazione, tra cui quella USB, con la quale è possibile rendere

facilmente fruibili i numeri casuali generati. Quindi, l'intero apparato comprende solo un microcontrollore e una minima catena analogica di elaborazione esterna e può essere interfacciato con estrema facilità ad elaboratori elettronici o sistemi embedded. La qualità della proposta, in termini di statistica delle sequenze casuali generate, è stata validata sfruttando i test standardizzati dall'U.S. National Institute of Standards and Technology.

Abstract

Is not possible to implement true random sources in digital hardware. Therefore, historically, it has been common to rely on pseudo-random number generators, thus avoiding the potential costs and long design times required by analog hardware. Unfortunately, pseudo-random sources have inherent vulnerabilities (reproducibility and periodicity) that make them hardly suitable for several computer security applications, such as cryptography. After some notable incidents caused by a misuse of pseudo-random generators, today the demand of true-random number generators is on the rise. Some major players in Information and Communication Technology have recently developed their own dedicated solutions, but these are only available on modern, high-end systems. Therefore, it is certainly timely to provide true-random generators for legacy or low-cost systems. To ensure security and at the same time a low implementation cost it is appropriate to think of architectures that can reuse ready available analog parts. In this respect some architectures based on chaotic dynamics are particularly interesting since that can base a large part of the analog processing chain on analog-to-digital converters. In fact, these blocks are widely available in integrated form, also as functional units of more complex systems such as, for instance, programmable circuits and microcontrollers.

In this paper, we propose a low cost and highly flexible design based on architecture deploying analog-to-digital converters, initially conceived at the University of Bologna. Cost reduction is achieved by exploiting the converter already integrated in a microcontroller. Flexibility comes from the fact that the chosen microcontroller provides a variety of communication interfaces, including a USB bus, which eases access to the random numbers generated on the device. Hence, the entire system includes only a microcontroller and a minimum of external components dedicated to signal processing and can be interfaced with extreme ease to computers or

embedded systems. The quality of the proposed system, in terms of statistics of generated random sequences, has been validated using standard test suite such as that defined by the U.S. National Institute of Standards and Technology.

Acronimi

RNG Random Number Generator

ISN Initial Sequence Number

TLS Transport Layer Security

SSL Secure Sockets Layer

VoIP Voice over IP

LCG Linear Congruential Generator

LFib Lagged Fibonacci generator

LFSR Linear Feedback Shift Register

PRNG Pseudo Random Number Generator

TRNG True Random Number Generator

HRNG Hardware Random Number Generator

SNR Signal to Noise Ratio

IID Indipendenti e Identicamente Distribuiti

PWAM Piece-Wise Affine Markov Maps

PDF funzione densità di probabilità

FPGA Field Programmable Gate Array

FPAA Field Programmable Analog Array

ADC Analog to Digital Converter

DAC Digital to Analog Converter

IS Invariant Set

INL Integral Non-Linearity

CRC Cyclic Redundancy Check

S/H Sample and Hold

BOM Bill Of Material

PGA Programmable Gain Amplifier

μC microcontrollore

μP microprocessore

I^2C Inter Integrated Circuit

SPI Serial Peripheral Interface

RAM Random Access Memory

MAPS Microchip®Advanced Part Selector

MSSP Master Synchronous Serial Port

GBP Gain Bandwidth Product

PCB Printed Circuit Board

CAD Computer Aided Desing

PLL Phase-Locked Loop

SMT Surface Mount Technology

CDC Communications Device Class

SO Sistema Operativo

LRNG Linux Random Number Generator

SHA Secure Hash Algorithm

TGSFR Twisted General Feedback Shift Register

RFC Request for Comments

NIST National Institute of Standard and Technology

FIPS Federal Information Processing Standards

STS Statistical Test Suite

Indice

Introduzione

Osservando il mondo che ci circonda si può notare un'infinità di eventi casuali. Questo può portare a pensare che sia relativamente semplice generare artificialmente eventi casuali. Tale conclusione sarebbe erronea. Indagando un po' più a fondo, è facile osservare che i sistemi di cui normalmente ci si serve per applicazioni di calcolo sono inadeguati a questo fine. In particolare, a causa della loro natura deterministica, per i sistemi digitali che pervadono la società moderna, generare numeri *autenticamente* casuali è del tutto impossibile. Paradossalmente, gli stessi sistemi di elaborazione hanno necessità crescente di buoni valori casuali, poiché essi trovano applicazione in moltissimi sistemi di sicurezza informatica e nei sistemi crittografici.

Tradizionalmente, i generatori di numeri casuali più utilizzati sono i cosiddetti Pseudo Random Number Generator (PRNG). Questi sono algoritmi in grado di espandere un seme o *seed* in una lunga sequenza irregolare di bit. Tuttavia, tale procedura è di per sé non casuale, poiché lega univocamente ogni bit al seme iniziale. In altre parole, l'unica sorgente di casualità è attribuile al seed.

La recente esplosione nel settore della sicurezza informatica ha portato a riconsiderare l'utilizzo indiscriminato dei PRNG. A tale scopo, grandi attori dell'industria informatica, come Intel® e VIA®, integrano, all'interno delle loro architetture più moderne, piattaforme sicure basate su True Random Number Generator (TRNG) invece che PRNG [28].

Convenzionalmente, i TRNG sono basati sull'osservazione esplicita di quantità analogiche relative a processi fisici complessi, come rumore termico, rumore di fase, fenomeni turbolenti o anche tempi di emissione del decadimento radioattivo. Tuttavia, tale approccio vincola il progettista alla tecnologia fisica sottostante. Inoltre, impiegando processi *naturalmente disponibili*, spesso il data rate risultante

è ridotto.

Al contrario delle altre implementazioni, il TRNG oggetto della proposta di tesi non è basato esplicitamente sull'osservazione di processi microcosmici complessi, bensì su primitive di *signal processing* standard, liberando il progettista dalla necessità di hardware dedicato. Infatti, il sistema sarà implementato partendo da blocchi di conversione analogico-digitale, oramai onnipresenti in architetture a microcontrollore o in sistemi embedded. Questo viene reso possibile grazie all'impiego di dinamiche caotiche.

I sistemi caotici o a dinamica complessa si collocano infatti al confine tra processi deterministici e stocastici. Costruendo un opportuno modello caotico artificiale è possibile trarre vantaggio da questa doppia natura e costruire un circuito in cui, con relativa facilità, si possono estrarre dallo stato del sistema sequenze binarie casuali. Grazie alla possibilità di scegliere il modello impiegato è possibile limitarsi a strutture dinamiche per le quali sono disponibili strumenti matematici di analisi. In questo modo sarà possibile descrivere il comportamento del sistema in termini statistici, sia in termini di distribuzione di probabilità che in termini di correlazione. Inoltre, una valida verifica sperimentale sarà fornita in ultimo luogo, grazie a batterie di test statistici standard, per provare la qualità e robustezza delle sequenze binarie casuali generate dal TRNG progettato.

Nel complesso, l'obiettivo di questo elaborato è l'implementazione di una generatore alternativo a quelli attualmente in commercio in grado di fornire numeri *autenticamente* casuali, allo scopo di colmare il divario tra PRNG e TRNG integrati nei sistemi più moderni. Beneficiari di questo sforzo sono soprattutto i *sistemi embedded*, i quali esibiscono sorgenti casuali limitate, a fronte di una sempre maggiore esigenza di sicurezza nelle comunicazioni in rete, requisito fondamentale nel processo di evoluzione del concetto di *Internet of Things* [29].

Capitolo 1

Applicazioni RNG nei sistemi informativi

Sequenze di bit casuali sono impiegate per svariati scopi all'interno dei sistemi crittografici. Le due applicazioni principali sono la generazione di chiavi o password e l'occultamento di dati sensibili. Queste due attività sono talmente importanti da far sí che la quasi totalità dei sistemi di sicurezza informatica dipenda dalla disponibilità di buoni Random Number Generator (RNG) per un corretto funzionamento. Inoltre, è necessario osservare che in un sistema crittografico composto da vari sottosistemi, il livello di sicurezza complessivo è in genere rappresentato dal livello di sicurezza dell'elemento più debole, così come la robustezza complessiva di una catena meccanica è rappresentata dalla robustezza della maglia più debole. Conseguentemente, l'utilizzo di RNG 'non sicuri' può compromettere un sistema crittografico, a prescindere dalla qualità con cui sono stati progettati i restanti blocchi. Un RNG è 'non sicuro' quando esiste il rischio, concreto o potenziale, che un'entità esterna possa prevederne le sequenze di uscita. La casualità è una risorsa fondamentale per la crittografia. Pertanto i RNG sono blocchi critici indispensabili in tutti i sistemi crittografici.

1.1 Utilizzatori di sequenze casuali

Nei sistemi informativi moderni possono essere evidenziate differenti classi di utilizzatori di sequenze casuali. In particolare, si può fare una distinzione in base

alle diverse esigenze:

- elevato throughput e ripetibilità: sistemi per simulazioni (es. metodo Monte Carlo);
- impredicibilità elevata: sistemi di sicurezza o estrazioni per giochi d'azzardo.

Consideriamo il primo caso, nel quale le sequenze casuali vengono utilizzate per effettuare simulazioni. Perché tali simulazioni siano accurate e veloci, è necessario disporre di un numero enorme di condizioni differenti e indipendenti tra loro, generate ad elevato throughput, al fine di ottenerne i risultati in tempi ridotti. La simulazione è spesso impiegata in ambiti (per esempio, nella ricerca scientifica o nei processi decisionali) in cui la riproducibilità dei risultati è importante. Quindi è utile avere sequenze con caratteristiche statistiche compatibili con eventi casuali, ma pienamente riproducibili.

Differentemente, nel secondo caso, il requisito fondamentale è quello dell'impredicibilità. Quasi tutti i sistemi crittografici si affidano alla nozione di *secret key*, la quale, per sua natura deve essere generata attraverso un metodo impredicibile. Se così non fosse, il sistema crittografico in sè sarebbe inutile.

Quindi, le due classi individuate risultano radicalmente diverse. Se da un lato si prediligono caratteristiche come quantità, velocità e riproducibilità, dall'altro lato l'unica proprietà cercata è quella dell'impredicibilità. Tale diversità deve essere tenuta in forte considerazione al momento della selezione del RNG per una particolare applicazione.

1.2 Sicurezza informatica

Come già accennato, nel settore della sicurezza informatica il requisito fondamentale per un RNG è quello dell'impredicibilità. Poiché i protocolli di sicurezza utilizzati in questo ambito si affidano all'impredicibilità delle chiavi impiegate, i RNG per applicazioni crittografiche devono soddisfare requisiti molto stringenti. Tra i più importanti emerge il seguente: una possibile entità ostile, anche conoscendo l'architettura del RNG, non deve essere in grado di fare qualsiasi tipo di previsione sullo stato futuro dell'uscita del generatore. Questo requisito è complesso da ottenere nella pratica e perfino più difficile da dimostrare in teoria. Anche

un RNG che genera numeri casuali, le cui valutazioni basate su sequenze finite mostrano buone caratteristiche statistiche, potrebbe non soddisfare il requisito. Sarà quindi compito del progettista identificare e sfruttare i soli processi che offrono migliori garanzie.

Nei sistemi informativi sono molteplici i componenti di sistema o di rete che necessitano di numeri casuali. Valutiamo, come esempio, quale utilizzo ne viene fatto nelle comunicazioni in rete su protocollo TCP/IP. I due esempi considerati sono: ordinamento dei pacchetti TCP e protocollo TLS.

Nelle comunicazioni TCP/IP, al momento della connessione, vengono generati gli Initial Sequence Number (ISN). Questi numeri di sequenza sono utilizzati nella comunicazione tra *client* e *server* per tenere traccia di ciascun pacchetto trasmesso e assicurare che il dialogo tra queste due entità abbia un corretto svolgimento. Recentemente è stata assodata la necessità che gli ISN siano valori impredicibili. È infatti noto che ISN predicibili possono causare un indebolimento del protocollo TCP, rendendo lo stesso vulnerabile ad attacchi di tipo *spoofing*[1], come evidenziato nel documento [18].

Ad un livello di astrazione più alto, il protocollo Transport Layer Security (TLS), successore del protocollo Secure Sockets Layer (SSL), è un protocollo di comunicazione progettato per fornire sicurezza nelle comunicazioni su reti TCP/IP. Nell'uso abituale (su applicazioni come browser, e-mail, messaggistica istantanea e VoIP) l'autenticazione TLS è unilaterale, ovvero è il solo *server* ad autenticarsi presso il *client*. Tale meccanismo è suddiviso in tre fasi:

- negoziazione per l'algoritmo da utilizzare;
- scambio delle chiavi di autenticazione;
- cifratura simmetrica e autenticazione dei messaggi.

Ognuna di queste fasi prevede lo scambio di informazioni in rete tra *client* e *server*, allo scopo di generare una chiave pubblica condivisa a entrambi. I messaggi scambiati tra i due soggetti saranno decifrabili solamente utilizzando la propria chiave privata, secondo uno schema crittografico asimmetrico [1]. È proprio la coppia di chiavi pubblica/privata a essere generata a partire da numeri casuali,

[1] Attacco informatico nel quale si attua una falsificazione d'identità.

utilizzando critto-algoritmi come RSA [46]. Inoltre, è noto che numeri casuali generati secondo algoritmi predicibili possono generare problemi di sicurezza, come riportato nella Sez. 2.2 del prossimo capitolo.

Questi, assieme ad altri esempi non citati, dovrebbero avere motivato il lettore a considerare la generazione di numeri casuali come un'entità fondamentale nei sistemi informativi.

Capitolo 2

Pseudo-RNG

I processi che avvengono all'interno di un elaboratore elettronico digitale sono perfettamente deterministici. Per questo motivo, generare numeri autenticamente casuali su tali sistemi è un compito impossibile. Tuttavia, esiste una famiglia di algoritmi, i Pseudo Random Number Generator (PRNG), in grado di produrre sequenze che nel breve termine approssimano efficacemente le proprietà dei numeri casuali, benché in realtà composte da campioni deterministicamente legati l'uno all'altro. Tali sequenze sono prodotte dall'espansione di un ridotto insieme di condizioni iniziali, il seme o *seed*. Conseguentemente, sistemi che condividono lo stesso generatore PRNG e lo stesso seed, separati nello spazio o nel tempo, generano sequenze di numeri casuali identiche tra loro. Quest'ultima proprietà, denominata ripetibilità, può risultare utile in alcune attività. Ad esempio, negli esperimenti scientifici basati su simulazione, la ripetibilità facilita la verifica da parte di diversi team di scienziati. Ancora, in certe applicazioni crittografiche, consente la generazione a distanza di chiavi di codifica condivise (es. nel sistema GPS o in alcuni sistemi di autenticazione). Tuttavia, è evidente che la ripetibilità è in conflitto con il concetto stesso di casualità. I PRNG sono comunque allettanti da un punto di vista implementativo anche per il loro elevato *data rate*, ovvero per la capacità di produrre elevate quantità di numeri casuali in un tempo ridotto.

Oltre che per la ripetibilità, i PRNG si caratterizzano anche per la loro *periodicità*. Infatti, ogni sistema di calcolo digitale è in ultima istanza una *macchina a stati*. Essendo finita la capacità di memoria utilizzabile da un PRNG quest'ultimo si caratterizza necessariamente come macchina a stati finiti. Inoltre, i PRNG sono

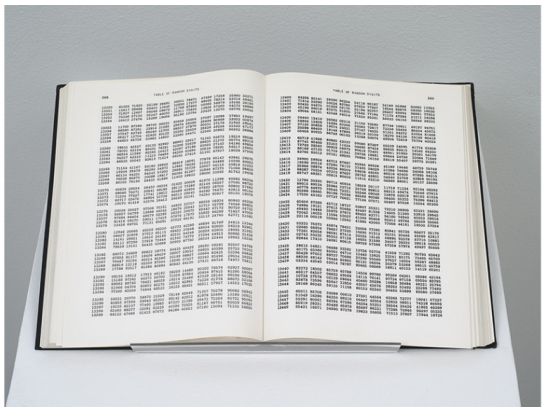


Figura 2.1: Foto del libro *A Million Random Digits with 100,000 Normal Deviates* [6]. (fonte [4])

privi di ingressi. Ogni volta che una macchina priva di ingressi viene a trovarsi in uno stato che ha già utilizzato, deve necessariamente sperimentare, da quel momento in poi, la medesima evoluzione di stato che ha attuato precedentemente. Si innesca cioè un comportamento periodico. Se la macchina è a stati finiti, ciò è inevitabile, perché la macchina non può continuare a utilizzare indefinitamente nuovi stati. Inoltre, la massima lunghezza di periodo che può realizzarsi è evidentemente data dal numero di stati disponibili. In alternativa, un sistema può produrre diversi cicli di periodo più breve. In questo caso, le condizioni iniziali determinano quale ciclo viene prodotto.

Nella pratica, la periodicità dei PRNG non è generalmente un problema. Infatti, è relativamente facile ottenere periodi così lunghi da inibire il manifestarsi della periodicità negli intervalli di normale utilizzo dei sistemi sui quali i PRNG sono implementati. Tuttavia, è evidente che periodicità troppo brevi potrebbero compromettere la sicurezza dell'applicazione. Conseguentemente, un buon algoritmo PRNG deve coniugare un elevato *throughput* con periodicità lunghe e buona semplicità implementativa.

2.1 Esempi di PRNG

Per meglio comprendere il funzionamento di un PRNG può essere conveniente fare riferimento al metodo utilizzato, prima dell'avvento dei calcolatori, nelle

simulazioni di esperimenti scientifici che esigevano valori casuali. A questo scopo si faceva riferimento a vere e proprie *raccolte* di valori, generate *una tantum* attraverso procedimenti fisici (es. estrazioni, lancio di dadi, misure su fenomeni turbolenti). La raccolta dei dati era soggetta a rigorosi controlli, con verifiche continue sulle statistiche dei dati raccolti. In tal modo, era possibile escludere qualunque polarizzazione o correlazione derivante da errori sistematici. Ad esempio, è particolarmente noto il libro [6], che contiene milioni di valori decimali casuali, divisi in pagine e ordinati in tabelle (si veda la Fig. 2.1).

Quando si aveva bisogno di numeri casuali, era sufficiente stabilire una pagina e una riga di partenza, utilizzando i valori letti da quel punto in poi. Se servivano più valori di quelli contenuti nel libro, arrivati alla fine della raccolta si poteva ripartire da capo, introducendo così una periodicità. Questo approccio è simile a quanto accade in un PRNG. Il punto di partenza costituisce un *seme* che attraverso la lettura viene espanso in una lunga sequenza. Il vantaggio dei PRNG è che consentono di risparmiare la memorizzazione dell'intera sequenza di valori, condensandola in una forma matematica.

Esistono svariate classi di algoritmi PRNG, tra cui:

- Linear Congruential Generator (LCG);
- Lagged Fibonacci generator (LFib);
- Linear Feedback Shift Register (LFSR).

Tutti tendono a fare riferimento a forme ricorsive del tipo

$$\begin{cases} \vec{x}(n+1) = \vec{f}(\vec{x}(n)) \\ u(n) = q(\vec{x}(n)) \end{cases} \tag{2.1}$$

Nell'espressione precedente, il vettore $\vec{x}$ rappresenta lo stato e la funzione $\vec{f}(\cdot)$ ne definisce l'evoluzione. La funzione $\vec{f}(\cdot)$ deve evidentemente essere non lineare. Il processo di assegnazione del seed corrisponde alla definizione di una condizione iniziale per $\vec{x}$. È possibile settare tutto $\vec{x}$ oppure solo una parte di esso, assumendo una configurazione fissa per la parte restante. La funzione $g(\cdot)$ produce i valori di uscita a partire dallo stato corrente. Il suo scopo è quello di impedire una piena osservazione dello stato del sistema. Conseguentemente, $g(\cdot)$ è sempre una

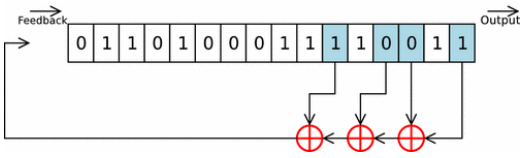


Figura 2.2: Funzionamento di un LFSR con polinomio $x^{16} + x^{14} + x^{13} + x^{11} + 1$: nello stato iniziale il registro contiene il valore del *seed*, pari a 0110100011110011.

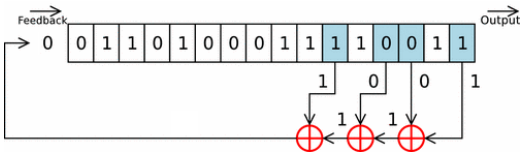


Figura 2.3: Funzionamento di un LFSR con polinomio $x^{16} + x^{14} + x^{13} + x^{11} + 1$: viene generato il bit di *feedback* operando uno XOR sui *tap* 11, 13, 14 e 16.

funzione non invertibile. Il tipo di generatore viene stabilito a partire dalla forma assunta da $\vec{f}(\cdot)$ e $g(\cdot)$.

A titolo di esempio, semplice ma concreto, è possibile evidenziare il funzionamento di un LFSR. Questa tipologia di PRNG implementa le funzioni $\vec{f}(\cdot)$ e $g(\cdot)$ a partire da due componenti elementari: *shift register* e *gate XOR*. Ad ogni passo computazionale, il contenuto di un registro a scorrimento viene fatto scorrere di una posizione. Come valore binario in ingresso viene utilizzato l'XOR dei valori contenuti in alcune posizioni prefissate del registro, denominate *tap*. Il bit in uscita è il valore binario casuale che è stato generato. La funzione di aggiornamento di stato è non lineare in termini assoluti, ma lineare nell'ambito di un'aritmetica dei campi finiti. Alla funzione di aggiornamento di stato di un LFSR può venire associato un polinomio caratteristico. Ad esempio, in Fig. 2.2 si ha un LFSR con polinomio caratteristico $x^{16} + x^{14} + x^{13} + x^{11} + 1$. Le potenze presenti nel polinomio indicano i *tap*, ovvero le posizioni del registro a scorrimento da cui, ad ogni passo, sono prelevati i valori che, combinati tramite XOR, determinano lo stato successivo. Il polinomio caratteristico rappresenta uno strumento matematico che consente di determinare alcune proprietà dell'LFSR, tra cui la possibilità di avere periodo massimo [14].

Osservando in sequenza le illustrazioni riportate, è evidente il funzionamento del sistema. Dallo stato iniziale in Fig. 2.2 si passa a quello di generazione in Fig. 2.3,

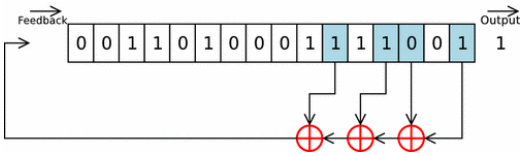


Figura 2.4: Funzionamento di un LFSR con polinomio $x^{16} + x^{14} + x^{13} + x^{11} + 1$: lo scorrimento del registro identifica il cambiamento nel nuovo stato 0011010001111001.

nel quale i *tap*, attraverso un'operazione di XOR, generano il bit di *feedback*, pari a 0.

A questo punto, operando uno scorrimento del registro, mostrato in Fig. 2.4, il sistema commuta nello stato successivo a quello iniziale. Ad ogni colpo di clock il sistema cambia stato, ripercorrendo le fasi di generazione e scorrimento. Se l'LFSR è massimale[2] produce una sequenza che passa attraverso tutti i possibili stati del registro, tranne quello che produce tutti zeri. Quindi, selezionando accuratamente il polinomio caratteristico, è possibile ottenere un LFSR in grado di produrre una sequenza di bit che possono essere ritenuti casuali. Inoltre, negli algoritmi più complessi, il periodo può raggiungere valori molto elevati, minimizzando la probabilità di osservare la stessa sequenza entro un numero finito di passi. Tuttavia, il sistema è descrivibile mediante una macchina a stati finiti; perciò, la lunghezza massima del periodo è vincolata al numero finito di bit utilizzati per codificare il generico stato del sistema.

Grazie a un vettore di stato binario e all'uso di operazioni booleane elementari, il LFSR rappresenta la tipologia di PRNG più semplice a cui si possa pensare. Esistono tuttavia PRNG molto più complessi. Un esempio significativo di PRNG più sofisticato, diventato un vero e proprio standard, è rappresentato dal *Mersenne twister* [34], di cui è nota l'implementazione MT 19937. L'esigenza di PRNG complessi nasce dalla volontà di ottenere periodi molto lunghi e soprattutto dal tentativo di superare alcuni problemi e *vulnerabilità* dettagliate nella sezione seguente. Ad esempio, l'MT 19937 offre un periodo di ripetizione enorme, pari a $2^{19937} - 1$, un'elevata velocità e l'assenza di correlazioni evidenti nelle sequenze prodotte. Nonostante questi sforzi, va tuttavia sottolineato che neanche i PRNG più

[2] Il numero di *tap* è pari e i loro valori sono numeri primi relativi.

complessi possono sfuggire alla loro natura di macchine a stati finiti. Conseguentemente, possono rendere le problematiche derivanti da tale natura meno evidenti o addirittura non percepibili con analisi praticabili in tempi e risorse ragionevoli allo stato delle conoscenze attuali. Non possono tuttavia eliminarle in assoluto né offrire garanzia assoluta che un'evoluzione delle conoscenze non possa portare a percepire difetti e vulnerabilità in maniera apprezzabile in futuro.

Dell'MT 19937 esistono implementazioni riutilizzabili, codificate sotto forma di librerie software per linguaggi di programmazione come Python, PHP e MATLAB. Sebbene l'algoritmo *Mersenne twister* sia molto più articolato, possiamo confrontarlo ancora con l'LFSR visto sopra, con il quale condivide in parte pregi e difetti. Infatti, può ancora essere descritto mediante una macchina a stati finiti periodica, deterministica e quindi intrinsecamente insicura.

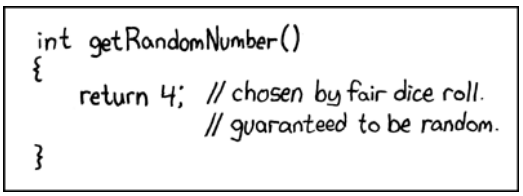
2.2 Problemi e vulnerabilità dei PRNG

La natura periodica dei PRNG fa sì che le sequenze prodotte presentino necessariamente delle correlazioni. Per i PRNG di peggiore qualità, le correlazioni possono risultare evidenti anche su scale temporali molto inferiori al periodo. Un modo pratico per evidenziare tali correlazioni consiste nel raggruppare consecutivamente i valori di uscita dei PRNG in modo da ottenere sottosequenze di valori interi. A titolo esemplificativo, si assuma che l'uscita di un PRNG sia utilizzata per produrre sequenze di M interi. A questo punto, i valori delle sottosequenze possono venire interpretati come coordinate di punti in uno spazio M-dimensionale. In pratica, ad ogni sottosequenza è possibile associare un punto nello spazio. Se sono presenti delle correlazioni, i punti possono non distribuirsi uniformemente nello spazio e ciò in maniera tanto più evidente quanto più M è grande. I PRNG più semplici, come i LFSR, tendono immediatamente a evidenziare distribuzioni non uniformi di punti anche per dimensionalità dello spazio M relativamente modeste. È evidente che la presenza di correlazioni così facilmente rilevabili può costituire un problema significativo in termini di applicabilità.

La presenza di correlazioni non è tuttavia il problema principale dei PRNG. Gli algoritmi più sofisticati consentono infatti di nascondere molto bene il fenomeno.

Ad esempio, l'algoritmo MT 19937 consente di generare punti perfettamente equidistribuiti in spazi fino a 623 dimensioni.

La problematica principale consiste nella possibilità potenziale, da parte di un osservatore, di *ricostruire* il vettore di stato del sistema. Per comprenderne la portata può essere utile riprendere l'esempio della raccolta di numeri casuali presentato precedentemente. Una siffatta raccolta è *equivalente* a un PRNG a patto di assumere come stato l'indice del valore corrente e come funzione di uscita la funzione che ritorna il valore puntato dall'indice. In questo parallelo, la funzione di evoluzione di stato consiste semplicemente nell'incremento dell'indice ad ogni passo. Un eventuale osservatore esterno, nota la raccolta utilizzata, a partire da una sottosequenza di uscita può immediatamente risalire all'indice corrente. Quindi, ottenuta tale informazione, può produrre in maniera autonoma gli stessi valori generati dal PRNG. In pratica non è garantita la *segretezza* dei valori futuri che saranno prodotti. Se anche vi fossero più raccolte, dall'osservazione di una sequenza di uscita, per un osservatore può essere possibile risalire sia alla raccolta sia all'indice. Nel caso di un PRNG, la raccolta è sostituita da un algoritmo e la funzione di stato è fortemente non invertibile. Ciò non garantisce tuttavia che un osservatore particolarmente accorto non possa riuscire, attraverso l'opportuna osservazione delle sequenze di uscita, a risalire ad algoritmo e stato. In molti casi, è impossibile tenere segreto l'algoritmo utilizzato, il che può semplificare il compito dell'osservatore. Inoltre, in alcune implementazioni l'osservatore potrebbe avere accesso (anche solo transitoriamente) a informazioni correlate con lo stato del sistema (*side channel*). Anche in questo caso il suo compito può venire semplificato. Quindi, qualunque evento dovesse consentire a un osservatore di ricostruire lo stato del sistema, offrirebbe allo stesso piena conoscenza di *tutti* i valori 'casuali' prodotti da quell'istante in poi.



La caduta della proprietà di *segretezza* dei valori casuali futuri, cioè dell'*impredicibilità* di un generatore, costituisce un grave *vulnus*, laddove il generatore

sia impiegato in sistemi di sicurezza informatica. Molti algoritmi crittografici fondano la propria sicurezza sull'uso di RNG e sul presupposto che le sequenze da essi prodotti siano impredicibili. La caduta di tale presupposto rappresenta quindi un'importante *vulnerabilità del sistema*.

Per comprendere quanto il problema sia concreto, è possibile fare riferimento a alcuni incidenti *di scuola*.

Le prime versioni del protocollo crittografico SSL, implementate nel browser Netscape®, utilizzavano algoritmi PRNG per la generazione di numeri pseudo-casuali. Malauguratamente, all'inizializzazione del sistema, il *seed* veniva generato a partire da dati facilmente accessibili, quali ora del giorno, *process-ID* e *parent process-ID*. Era quindi facile per un'entità ostile ricostruire lo stato del PRNG e, da quel momento in poi, garantirsi la possibilità di accedere alle informazioni crittografate. Purtroppo, il problema, identificato nel 1994 da Phillip Hallam-Baker, non fu risolto prima del rilascio del browser. Solamente nel 1995, due studenti del dipartimento di computer science della University of California (Berkeley), Ian Goldberg e David Wagner, evidenziarono il *bug* nell'applicazione [13]. Ciò richiese un lavoro di *reverse engineering* del codice, in quanto Netscape® si era rifiutata di rilasciare i dettagli relativi al suo generatore di numeri casuali. A seguito di questa scoperta, l'implementazione del PRNG venne corretta, rendendo più robusto il meccanismo di *seeding*.

Un altro esempio noto è quello relativo all'attacco Java™ *session-ID* [15]. Il problema, rilevato da Zvi Gutterman e Dahlia Malkhi, riguardava i servlet[3] Java™ usati in applicazioni di E-commerce. In particolare, a essere vulnerabile era la generazione del valore a 128 bit del *session-ID*, su cui si basavano le transazioni. Anche in questo caso, i ricercatori hanno dimostrato la vulnerabilità del PRNG impiegato, attaccabile in 2^{64} iterazioni o anche meno.

Infine, riportiamo un evento più recente, riguardante la rete Bitcoin[4] [43]. Nel 2013, un problema nell'implementazione del PRNG di Java™ SecureRandom in ambiente Android™, ha reso possibile furti di migliaia di *bitcoin*, producendone una rapida svalutazione.

[3] Codici in linguaggio Java™ che operano all'interno di un web server permettendo la creazione di applicazioni web.

[4] Il *bitcoin* è una moneta elettronica, creata nel 2009. Aspetto saliente di questa valuta è l'uso di procedimenti crittografici per la generazione e il trasferimento di moneta.

Come conclusione, va osservato che esistono classi di PRNG per le quali sono stati formalizzati metodi che in tempi ridotti consentono di risalire allo stato del sistema a partire dalla semplice osservazione di sottosequenze di uscita. Per esempio è il caso dei LFSR. Per i PRNG più complessi, ad oggi non vi sono metodi noti. Tuttavia, ciò non elimina il problema. Infatti, non è detto che in un prossimo futuro non vi siano scoperte tali da consentire di formalizzare metodi di ricostruzione di stato anche per PRNG per i quali, ad oggi, tali metodi non esistono. Inoltre, non si può escludere la presenza di *side channel*, i quali possono favorire la ricostruzione dello stato da parte di un'entità ostile. Il problema della predicibilità dei PRNG è quindi trasversale a tutta questa classe di RNG e non legato a un particolare algoritmo o a una particolare implementazione.

Capitolo 3

True-RNG

I True Random Number Generator (TRNG) sono generatori di numeri casuali in grado di offrire caratteristiche di *autentica impredicibilità*. Come si è visto nel Cap. 2, non essendo possibile ottenere tale risultato tramite algoritmi, i TRNG fanno necessariamente riferimento ad architetture *hardware* dedicate. Per questa ragione spesso viene usato il termine Hardware Random Number Generator (HRNG)[5].

Essendo qualsiasi sistema hardware digitale una macchina a stati finiti, è evidente che i generatori TRNG dovranno comprendere parti analogiche o essere dotati di ingressi che attingono in ultima istanza da sorgenti analogiche. Infatti, per svolgere il proprio compito, i TRNG devono necessariamente fare riferimento a fenomeni non deterministici di natura fisica. I processi fisici su cui si basano si distinguono per la loro caratteristica principale: l'*impredicibilità*. Tale particolarità è intimamente legata alla struttura microscopica dei processi fisici interessati e a una forte dipendenza dalle condizioni iniziali. Inoltre, la teoria dei sistemi dinamici instabili e la teoria del caos (vedi Capitolo 4) forniscono validi strumenti matematici per giustificare la presenza di impredicibilità in questi sistemi.

Tipicamente un True-RNG è costituito da un trasduttore o meccanismo di acquisizione, in grado di convertire una grandezza fisica macroscopica, nota per essere influenzata da una pluralità di fenomeni microcosmici incontrollabili, in un segnale elettrico, a partire dal quale viene prodotto un valore numerico. Considerando più acquisizioni successive, si ottengono sequenze casuali di bit. Sebbene il flusso di

[5]Tale terminologia può però dare luogo ad ambiguità, in quanto anche un PRNG può essere implementato in hardware.

bit così prodotto possieda intrinsecamente ottime caratteristiche statistiche, questo potrebbe contenere ancora deboli correlazioni e squilibri nella distribuzione (bias). Per questo motivo i TRNG solitamente comprendono un meccanismo di post-processing, atto a eliminare le correlazioni residue, rimuovere il bias e aumentare la qualità generale della sequenza prodotta, come mostrato in seguito in Sez. 5.4.1.

Esistono anche RNG *ibridi*, i quali uniscono caratteristiche dei PRNG a quelle dei TRNG. Infatti, nel caso in cui il tasso di produzione di numeri casuali in uscita non sia soddisfacente, è possibile impiegare periodicamente i numeri prodotti dal TRNG come seed di un PRNG, il quale, espandendo tale seme, produce un flusso di numeri pseudo casuali a elevato data rate. Basandosi su processi intrinsecamente casuali, i TRNG sono privi dei problemi di sicurezza dei PRNG, evidenziati nel Cap. 2. Tuttavia, se comparati in termini di tasso di produzione di numeri casuali, i TRNG sono in media più lenti rispetto ai PRNG. Quindi, in questi casi particolari, dove è richiesto un throughput elevato, si preferisce impiegare i RNG *ibridi*.

3.1 Sorgenti di rumore per TRNG

Esempi comuni di generatori di numeri casuali sono il lancio di una moneta, i dadi, la roulette o l'estrazione della lotteria. Sebbene questi ultimi si basino su processi macroscopici (ad esempio le due facce di una moneta), le perturbazioni che alterano il risultato di ogni esperimento sono innumerevoli e microscopiche. In aggiunta, un'infinità di condizioni iniziali concorrono a identificare lo stato iniziale del sistema, rendendo ogni esperimento impredicibile e indipendente. Gli esempi appena citati, oltre ad avere un data rate molto ridotto, sono del tutto inutilizzabili nei moderni sistemi informativi, come i computer o i sistemi embedded. Piattaforme di questo tipo prediligono HRNG, di cui nel seguito sono elencate alcune possibili implementazioni, basate su sorgenti di rumore differenti.

3.1.1 Attività umana

Numeri casuali possono essere prodotti direttamente da attività umane. Molti studi hanno dimostrato che le attività umane, soprattutto quando risultano collegabili a grandi comunità di soggetti o sono influenzate dall'ambiente, possiedono

un certo grado di casualità. Nei sistemi di calcolo è possibile trarre vantaggio da tale circostanza ad esempio raccogliendo informazioni come sequenze di lettere digitate su una tastiera, intervalli di tempo tra le pressioni dei tasti, movimenti di periferiche di comunicazione uomo-macchina (come il mouse) [58]. Tuttavia, la variabilità sistematica rilevata negli esperimenti tra diversi individui, porta a scartare l'attività umana come sorgente di rumore se comparata a buoni generatori casuali.

Nei sistemi che non dispongono di altre fonti di rumore, le attività umane sono spesso impiegate per generare i *seed* di PRNG o per alimentare 'entropy buffer' dai quali vengono 'distillati' valori casuali. La distillazione viene attuata impiegando trasformazioni che praticano un intenso mescolamento tra i valori disponibili e che tipicamente comprimono la quantità di dati, comportando un volume di valori casuali estraibile in uscita minore di quello immesso. Questo è il caso, ad esempio, dell'entropy pool di Linux™, il quale assume come sorgenti la digitazione su tastiera e il movimento del mouse.

3.1.2 Fenomeni rumorosi in dispositivi elettronici

I fenomeni rumorosi elettronici fanno parte di una classe di fenomeni fisici. Impiegare tali processi nella generazione di numeri casuali è relativamente agevole, in quanto la stragrande maggioranza delle architetture hardware fa riferimento al dominio elettronico. Di seguito, riportiamo alcuni fenomeni rumorosi elettronici noti in letteratura, i quali possono essere sfruttati come sorgenti di rumore nei TRNG.

3.1.2.1 Rumore termico

Il rumore termico, anche detto rumore Johnson-Nyquist, è generato dall'agitazione termica dei portatori di carica (elettroni) all'interno di un conduttore elettrico in situazione di equilibrio. Il rumore prodotto è pressoché bianco, ovvero la densità spettrale di potenza è costante per ogni frequenza, solo quando il resistore è ideale. Nel caso di un resistore reale, limitando l'osservazione ad una banda finita

di frequenze, si ottiene un andamento non uniforme, tipicamente Gaussiano. La densità spettrale di potenza è

$$\bar{v_n^2} = 4k_BTR \tag{3.1}$$

dove k_B è la costante di Boltzman, T è la temperatura assoluta del resistore (in K) e R è la resistenza (in Ω).

Amplificando opportunamente la caduta di tensione ai capi di un resistore reale, si realizza un generatore casuale di tensione. Sebbene l'implementazione di un generatore di questo tipo sia triviale, alcuni aspetti chiave lo rendono insicuro. Infatti, essendo basato su un processo fisico strettamente dipendente dalla temperatura (vedi la 3.1), è vulnerabile ad attacchi nei quali quest'ultima viene ridotta, limitando così il livello di impredicibilità. Inoltre, risulta complessa la discriminazione di un segnale interferente esterno rispetto al solo rumore termico, con la conseguente possibilità di compromettere la sicurezza del generatore.

3.1.2.2 Rumore shot

In elettronica il rumore shot, anche detto rumore Schottky, è il rumore prodotto dai portatori di carica quando attraversano una barriera di potenziale, presente, ad esempio, nei pressi di una giunzione *p-n*. La natura intrinsecamente discreta della carica elettrica permette di caratterizzare il fenomeno come un processo di Poisson, che descrive l'occorrenza di eventi casuali indipendenti. Se il numero di eventi considerati è elevato, la distribuzione di Poisson viene approssimata a una distribuzione Gaussiana, rendendo indistinguibile il rumore shot dal rumore Gaussiano[6] (rumore termico).

Il rapporto tra segnale e rumore (SNR) di questo processo è intimamente legato al numero di eventi considerati N, in quanto $SNR = \sqrt{N}$. Tuttavia, il rumore shot può facilmente essere dominato da altre sorgenti di rumore, come quello termico, quando N è molto elevato. Siccome lo spettro di potenza del rumore shot ha un andamento fortemente piccato, è necessario un adeguato filtraggio per poter

[6] Avente funzione densità di probabilità normale $p_G(z) = \frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-\frac{(z-\mu)^2}{2\sigma^2}}$, dove μ è il valore medio e σ^2 la varianza.

ottenere uno spettro piatto, utile per evitare correlazioni evidenti nelle sequenze prodotte.

Per evidenziare il comportamento del solo rumore shot è indispensabile mantenere costanti i contributi dovuti ad altre sorgenti di rumore. Perciò, TRNG basati su questo tipo di rumore devono tenere in conto la possibilità concreta di attacchi esterni, atti a indebolire il generatore, mediante i quali si punta alla dominazione del rumore shot con sorgenti di rumore maggiormente predicibili.

3.1.2.3 Rumore flicker

Il rumore flicker, anche conosciuto come rumore $1/f$, è un processo caratterizzato da un spettro tipicamente rosa, il quale ha un andamento costante decrescente ad elevate frequenze. Sempre presente nei dispositivi elettronici, ha rilevanza maggiore a basse frequenze, in quanto ad alte frequenze viene oscurato dal rumore bianco. Negli oscillatori questo rumore a bassa frequenza viene modulato a frequenze nell'intorno della portante, tramutandosi così in rumore di fase.

La presenza di rumore $1/f$ può essere dovuta ad una varietà di effetti, quali impurità nel canale conduttivo (nei dispositivi MOS) o processi di generazione-ricombinazione (nei dispositivi BJT). Anche dispositivi passivi, come i resistori standard a strato di carbone, sono fonte di elevato rumore flicker, il quale si aggiunge al rumore termico presente in tutti i resistori.

TRNG basati sul rumore flicker possono sfruttare sorgenti come la caduta di potenziale ai capi di un resistore rumoroso (vedi 3.1.2.1) o il jitter di fase che affligge gli oscillatori [31]. Sotto l'aspetto della sicurezza del generatore valgono considerazioni analoghe a quelle evidenziate per il rumore shot (vedi 3.1.2.2).

3.1.2.4 Rumore per effetto valanga

L'effetto valanga è un processo di moltiplicazione della corrente che insorge in materiali semiconduttori ed isolanti sotto l'effetto di elevati campi elettrici. Soprattutto nei dispositivi a semiconduttore, tali campi elettrici derivano dalla presenza di voltaggi moderati applicati a strutture di dimensioni infinitesime. Il ridotto numero di elettroni liberi presenti a temperatura ambiente in tali materiali vengono accelerati dal campo elettrico fino al punto di produrre nuovi elettroni ionizzati,

strappati dagli atomi della struttura. L'evoluzione segue una legge esponenziale creando un effetto a valanga. Questo processo fisico ha luogo nei diodi a valanga, nei diodi Zener o nelle giunzioni *p-n* polarizzate in inversa (in prossimità della regione di *breakdown*), nei quali causa un flusso di corrente affetto da rumore.

I TRNG basati sul rumore a valanga devono effettuare inizialmente una conversione corrente-tensione, seguita poi da una conversione analogico-digitale. Siccome le proprietà statistiche dei dati binari in uscita dall'Analog to Digital Converter (ADC) sono imperfette, è indispensabile aggiungere una fase di post-processing.

In commercio sono già disponibili diverse soluzioni TRNG che sfruttano questo tipo di rumore. Queste esibiscono buone qualità sia in termini di casualità, che in termini di costi ridotti, in quanto impiegano elettronica discreta a basso costo.

3.1.3 Fenomeni rumorosi in dispositivi fisici

Ovviamente, quello visto nella sezione precedente è il caso più conveniente, in quanto l'informazione casuale è già a disposizione del dominio elettronico, senza la necessità di meccanismi di conversione tra differenti domini (es. ottico-elettronico). Tuttavia, non mancano casi in cui si fa riferimento a fenomeni fisici di diversa natura.

3.1.3.1 Processo fotonico

Il fotone è il pacchetto elementare di energia che costituisce la radiazione elettromagnetica, come, per esempio, un fascio di luce. Questa discretizzazione della luce in elementi particellari permette di costruire dispositivi fotonici in grado di produrre numeri casuali a elevato data rate. In particolare, esponiamo le soluzioni fotoniche di QUANTIS® [52] e qStream [44].

I fotoni, prodotti da una sorgente coerente (laser), sono indirizzati verso uno specchio semi-trasparente, all'interno di una camera sottovuoto. Lo specchio divide il flusso luminoso in due percorsi distinti - uno riflesso e uno trasmesso - i quali terminano in due rivelatori ottico-elettronici (vedi Fig. 3.1). Essendo i fotoni particelle discrete, l'evento legato alla riflessione/trasmissione ha proprietà esclusive, a cui possiamo associare i valori binari '0' e '1', equiprobabili se lo

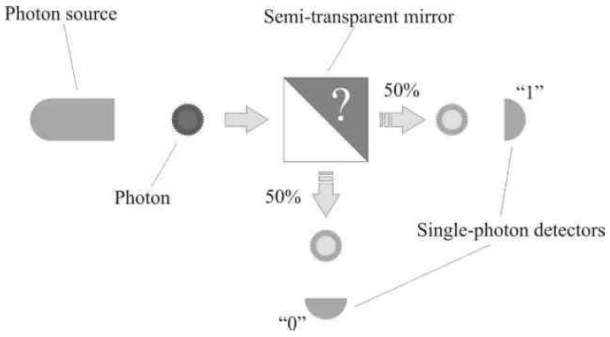


Figura 3.1: Schema a blocchi del sistema fotonico QUANTIS®per la generazione di numeri casuali. (fonte [52])

specchio ha un tasso di riflessione pari al 50%. Siccome la statistica così prodotta ha andamento Gaussiano, è necessario attuare un condizionamento in uscita, in modo tale da ottenere una sequenza *random* uniformemente distribuita e priva di *bias* [32].

In dispositivi fotonici è necessario prestare particolare attenzione soprattutto al processo di conversione da mondo ottico a elettrico: il rumore ottico deve prevalere sul rumore elettronico. Se così non fosse, le proprietà statistiche delle sequenze generate sarebbero compromesse e non deriverebbero unicamente da fenomeni ottici.

Sebbene il data rate prodotto in uscita sia molto elevato (nell'ordine dei Gbps), questo tipo di TRNG richiede tecnologie e metodi produttivi all'avanguardia, i quali incidono fortemente sul costo finale del dispositivo. Viste le ultime considerazioni, è facile intuire come RNG di questo genere siano usufruibili principalmente su sistemi di calcolo complessi, mentre non siano idonei, in termini di rapporto costo/prestazioni, se adottati su sistemi embedded e piccole piattaforme informative.

3.1.3.2 Decadimento radioattivo

Il decadimento radioattivo è il processo nel quale un nucleo di un atomo instabile perde energia emettendo radiazione. Tale processo ha caratteristiche stocastiche casuali a livello dei singoli atomi, quindi è impossibile predire l'esatto momento in cui un atomo decaderà. Siccome il livello di impredicibilità è elevato,

grazie alle considerazioni della meccanica quantistica, un TRNG basato su questa tecnologia può essere considerato quasi perfetto.

L'acquisizione elettronica di emissioni radioattive è affidata a un contatore Geiger, il quale rileva emissioni nucleari - particelle alfa, beta e raggi gamma - e produce un segnale elettrico a logica binaria. TRNG basati su questa tecnologia trovano impiego in campi limitati, in quanto valgono nuovamente considerazioni analoghe a quelle già viste nella Sez. 3.1.3.1.

3.1.3.3 Acquisizione di un sensore

L'acquisizione di fenomeni di natura fisica può avvenire anche per mezzo di sensori relativamente standard o di uso comune. Per esempio, la registrazione digitale di un'immagine da parte di una webcam o di una fonte sonora attraverso una scheda audio possono essere generalizzati, insieme ad altri esempi non citati, ad acquisizioni di un sensore. Se i fenomeni macroscopici sotto osservazione tendono ad essere caotici, le informazioni provenienti da ciascuno di essi possono essere opportunamente combinate, filtrate o venire utilizzate per inizializzare algoritmi PRNG [10].

Un gruppo di ricerca ha teorizzato la registrazione video di una *lava lamp*[7] per la generazione di numeri casuali [35]. Altri esempi di soggetti video potrebbero essere i movimenti delle bolle d'aria in un acquario o nastri agitati dal flusso d'aria prodotto da una ventola.

Purtroppo, il problema fondamentale legato ad ognuno di questi processi macroscopici è l'impossibilità di determinare la reale casualità del fenomeno osservato. Inoltre, l'immagine digitalizzata generalmente contiene rumore aggiuntivo, quindi non del tutto casuale, come quello legato alla conversione analogico-digitale. Un'altra sorgente digitale disponibile su quasi tutte le piattaforme è la scheda audio. Il rumore proveniente dall'ingresso audio, lasciato scollegato, può essere acquisito ed utilizzato per generare numeri casuali [57]. Vista però la possibilità di correlazioni e interferenze esterne, il flusso binario deve essere adeguatamente processato via software. In linea con quanto appena detto, su sistemi embedded

[7] Una lampada contraddistinta da un contenitore di vetro, all'interno del quale fluttuano liberamente bolle di cera liquida sospese in un liquido trasparente.

o a microcontrollore è verosimile acquisire informazioni caotiche da uno degli ingressi analogici disponibili, quando lasciato flottante [30].

Se da un lato il costo implementativo legato a questi TRNG sia praticamente nullo, in quanto viene sfruttato hardware già preesistente, dall'altro, l'inevitabile post-processing ne riduce notevolmente il *throughput* in uscita. Inoltre, la difficoltà nel garantirsi contro interventi esterni atti a modificare il comportamento dello stesso generatore, costituisce un limite fondamentale.

3.1.4 Meccanismi di fruizione

In alcuni casi, non è possibile fare riferimento diretto al fenomeno fisico di basso livello. Questa impossibilità può essere dovuta a un'inadeguatezza delle caratteristiche dello stesso processo. La stessa operazione potrebbe offrire a un eventuale attività ostile modalità semplici per sostituirsi al fenomeno fisico utilizzato. Conseguentemente, la maggior parte dei TRNG impiega meccanismi di fruizione dei fenomeni fisici impredicibili, i quali realizzano trasformazioni e combinazioni di fenomeni diversi.

3.1.4.1 Rumore di fase

Il rumore di fase è causato da instabilità nel dominio dei tempi, dette anche *jitter*, le quali provocano fluttuazioni casuali della fase di una forma d'onda nel dominio delle frequenze. Negli oscillatori l'incertezza legata alla determinazione della fase deve essere limitata secondo i vincoli di progetto, mentre può essere sfruttata a nostro favore quando si parla di TRNG. Ogni oscillatore, per definizione non lineare, possiede cicli limiti stabili, i quali vengono rappresentati da traiettorie chiuse nello spazio degli stati. Ogni perturbazione si trasforma in una deriva sulla fase, producendo un rumore di tipo ciclostazionario. In tali condizioni, il campionamento reciproco di più oscillatori può essere interpretato come un preliminare processo di post-processing, intrinseco nella natura stessa del rumore di fase, nel quale lo stato del sistema viene confinato entro limiti finiti dello spazio degli stati. Per questo motivo la statistica prodotta da questo tipo di sistema ha un andamento uniforme, perfetto per la realizzazione di TRNG.

Allo scopo di sfruttare il rumore di fase vengono realizzate batterie di oscillatori (minimo due), i quali si campionano vicendevolmente. In particolare, il caso tipico è quello in cui un oscillatore a bassa frequenza, quindi lento, campiona un oscillatore ad elevata frequenza, più veloce. Esempi simili sono presenti nelle architetture di Intel® e VIA®, che saranno analizzate nel dettaglio in seguito nella Sez. 3.2.1.

3.1.4.2 Sistemi a dinamica complessa

I sistemi a dinamica complessa o caotici sono il fulcro di questo elaborato e la loro trattazione verrà esplicitata nel dettaglio nel Cap. 4. Quindi, tratteremo la teoria matematica necessaria all'analisi di tali sistemi, allo scopo di sfruttarli per la generazione di numeri casuali.

3.2 Esempi di piattaforme TRNG esistenti

Prima di introdurre la nostra proposta di TRNG basato su dinamica caotica è bene identificare i TRNG già presenti sul mercato. Così facendo sarà possibile effettuare comparazioni in ambito di prestazioni, consumi e costi. Vediamo di seguito qualche esempio di piattaforme per la generazione di numeri casuali.

3.2.1 TRNG integrati su chipset e μP

Vista la progressiva richiesta di sicurezza in ambito *computing*, importanti attori del mondo informatico hanno iniziato a integrare TRNG all'interno dei processori di ultima generazione.

Consideriamo il caso di Intel® e, in particolare, al funzionamento dell'Intel® Random Number Generator, integrato nei processori della famiglia *Ivy Bridge* [17], il quale è schematizzabile (vedi Fig. 3.2) in tre blocchi distinti:

- una sorgente di rumore, combinazione di rumore termico, flicker e shot. Per ottenere questa sorgente vengono amplificate ed in seguito campionate le tensioni ai capi di due resistori adiacenti. La vicinanza tra questi ultimi garantisce che siano soggetti alle stesse condizioni di lavoro; in questo modo, calcolando la differenza tra i due segnali, si riducono eventuali correlazioni non imputabili direttamente al rumore.

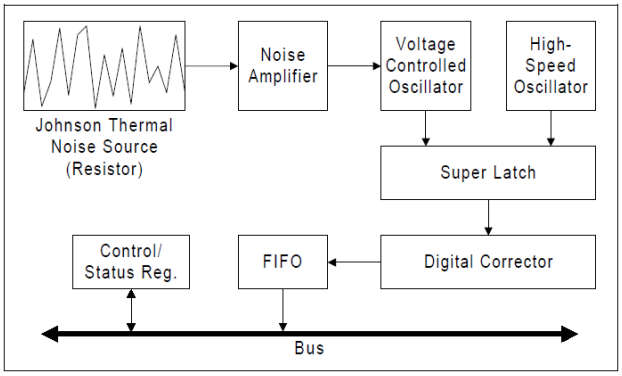


Figura 3.2: Schema a blocchi della sorgente di rumore del RNG Intel®. (fonte [28])

Inputs	Output
0,0	none
0,1	1
1,0	0
1,1	none

Tabella 3.1: Tabella di verità del correttore Von Neumann.

- un'architettura a due oscillatori, nella quale un oscillatore lento ($f_{LS} \approx 0.8\,\mathrm{GHz}$) produce il campionamento di un oscillatore più veloce ($f_{HS} \approx 3\,\mathrm{GHz}$). L'oscillatore lento è modulato in frequenza dalla sorgente di rumore. Il *drift* tra le due frequenze degli oscillatori genera bit casuali. È evidente che tale meccanismo è equivalente al rumore di fase.

- il post-processing, riceve in ingresso il flusso ad elevato data rate proveniente dall'architettura a due oscillatori, producendo in uscita un flusso a data rate variabile di bit casuali, bilanciato e privo di *bias* residui. Il correttore si basa sull'algoritmo proposto da John von Neumann [37], applicato a due bit consecutivi in ingresso (vedi Tab. 3.1).

La proposta di Intel® esibisce elevate performance sia in termini di data rate (superiore a $75\ \mathrm{Kbit/s}$ all'uscita del correttore) che in termini di casualità della se-

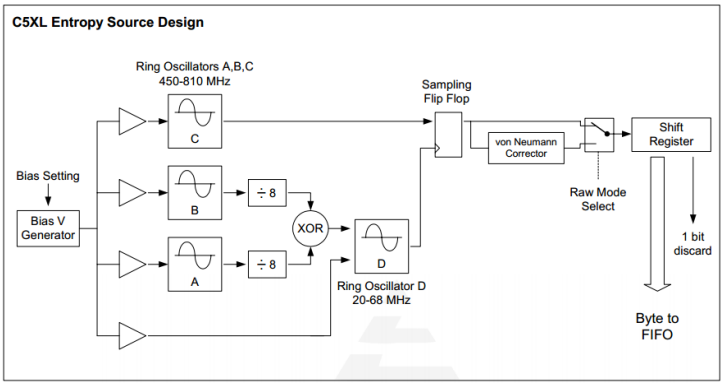


Figura 3.3: Schema a blocchi della sorgente di rumore del RNG VIA® C3 Nehemiah. (fonte [45])

quenza binaria prodotta, passando i maggiori test statistici sotto svariate condizioni di lavoro [28].

Anche VIA®Technologies, altro colosso informatico, propone una soluzione simile a Intel®. Infatti, l'RNG integrato nei processori C3 Nehemiah®di ultima generazione sfrutta ancora il rumore di fase. Il generatore può essere schematizzato in due blocchi distinti:

- un'architettura a quattro oscillatori, tre dei quali - A, B e C - sono oscillatori identici veloci ($f_{A,B,C} = 415 \div 810\,\mathrm{MHz}$), mentre il quarto - D - è più lento ($f_D = 20 \div 68\,\mathrm{MHz}$). I segnali di uscita di A e B sono miscelati mediante uno XOR e generano il campionamento dell'oscillatore lento D. Quest'ultimo produce, a sua volta, il campionamento del segnale proveniente dall'oscillatore C, generando un flusso di bit casuali ad elevato data rate. Il *jitter* tra le frequenze degli oscillatori è dipendente da rumore termico, variazioni nella produzione, temperatura e condizioni di lavoro.

- il post-processing, il quale riceve in ingresso il flusso ad elevato data rate proveniente dall'architettura a quattro oscillatori. Le proprietà statistiche di questo stream sono però imperfette. Per questo è possibile convogliare tale stream in un correttore Von Neumann (vedi Tab. 3.1), producendo in uscita un flusso a data rate variabile di bit casuali, bilanciato e privo di *bias* residui.

La proposta di VIA® esibisce eccezionali performance in termini di data rate ($4 \div 9\,\mathrm{Mbit/s}$ all'uscita del correttore e $30 \div 50\,\mathrm{Mbit/s}$ all'uscita *raw*). Ottimi

risultati anche in termini di casualità della sequenza prodotta, passando i maggiori test statistici in condizioni di lavoro estreme [45].

3.2.2 Piattaforme TRNG per *retrofit*

Come riportato nella sezione precedente, le piattaforme più moderne integrano già al loro interno RNG. Tuttavia, non mancano casi in cui tali risorse non siano disponibili, a causa, ad esempio, di limitazioni economiche o implementative. Una possibile soluzione a questi problemi è il *retrofit*. Il *retrofit* consiste nell'aggiungere nuove tecnologie o funzionalità a un sistema vecchio, prolungandone così la vita utile. Ovviamente, nel caso particolare, la nuova tecnologia è rappresentata dal RNG, mentre le funzionalità aggiuntive sono quelle di generazione ed approvvigionamento di numeri casuali. Grazie al retrofit è possibile aumentare il livello di sicurezza su sistemi datati oppure garantirla su sistemi altrimenti insicuri. Quest'ultimo è il caso, ad esempio, di alcuni sistemi embedded, i quali, come vedremo nella Sez. 5.5.1, dispongono di sorgenti casuali limitate.

I dispositivi TRNG per retrofit presenti sul mercato sfruttano svariate sorgenti di rumore, come quelle sopracitate. A seconda della complessità dell'architettura e delle prestazioni, il prezzo può variare notevolmente (da qualche decina fino a migliaia di dollari). Allo scopo di identificare parametri di confronto tecnologici ed economici tra la proposta discussa in questa tesi e TRNG commerciali, consideriamo le due soluzioni EntropyKey e NeuG.

Ora, consideriamo il generatore Entropy Key prodotto da Simtec Electronics [9]. Questo HRNG sfrutta il rumore per effetto a valanga generato da una giunzione *p-n* polarizzata in inversa, come già descritto nella Sez. 3.1.2.4. Al centro della sua architettura troviamo un processore ARM® Cortex™-M3 a $72\,\mathrm{MHz}$, in grado di campionare ad elevata frequenza la sorgente di rumore. La generazione di numeri casuali segue un'architettura articolata, nella quale è integrato il *self test* FIPS 140-2 (vedi Cap. 6). La comunicazione con il computer *host* avviene per mezzo dello standard USB Communications Device Class (CDC), attraverso il quale viene emulata una porta seriale virtuale. Il prezzo abbordabile (vedi Tab. 3.2) e le prestazioni elevate rendono Entropy Key un valido competitor. Tuttavia, la

disponibilità di questo TRNG è limitata, in quanto l'azienda produttrice sembra averne sospeso la produzione.

Come secondo esempio per *retrofit*, esaminiamo il generatore NeuG prodotto da Flying Stone Technology [56]. La piattaforma, anche denominata FST-01, ha caratteristiche similari alla soluzione precedente. Infatti, NeuG e EntropyKey condividono lo stesso processore e il protocollo di comunicazione. Quello che contraddistingue l'FST-01 è la sorgente di rumore. A tale scopo vengono utilizzate le sequenze digitali prodotte dai due convertitori A/D integrati nel microprocessore (μP), perturbate da svariati fenomeni rumorosi. In particolare, vengono adoperate le sequenze prodotte dall'acquisizione di:

- tensione di riferimento integrata;
- tensione del sensore di temperatura integrato;
- due tensioni di alimentazione.

I dati così acquisiti vengono miscelati da una funzione Cyclic Redundancy Check (CRC) a 32 bit ed inviati ad un algoritmo di *hash*, del tipo SHA-256. L'ultimo passaggio è cruciale in quanto permette di operare un post-processing aggressivo sulla sequenza in uscita. Tuttavia, l'implementazione di tale algoritmo è complessa, se confrontata con algoritmi più elementari (es. estrattore di Von Neumann nella prossima sezione). Anche in questo caso, il prezzo più che abbordabile (vedi Tab. 3.2) e il throughput elevato, pari a $70\,\mathrm{KB/s}$, rendono NeuG un validissimo competitor. Inoltre, il progetto è totalmente *open source*.

3.3 TRNG Integrati, Retrofit ed Embedded System

Il tema della sicurezza informatica ha avuto uno sviluppo esponenziale in questi anni, divenendo fondamentale in settori nei quali, in precedenza, era considerato secondario. Per questo motivo, sistemi informativi datati hanno oggigiorno la necessità di incrementare i propri livelli di sicurezza, rispetto agli standard di un tempo.

La soluzione adottata da Intel® e VIA® permette di rendere più sicuri i sistemi informativi recenti. Sebbene questa scelta fornisce da un lato piena integrazione

e ottime prestazioni, dall'altro lato è necessario tenere in considerazione le caratteristiche che l'architettura deve possedere. Infatti, essendo un'evoluzione dei processori di ultima generazione, solo questi ultimi possono beneficiare dei TRNG integrati.

Nei sistemi che non possono beneficiare di questa nuova tecnologia è possibile adottare un *retrofit*, aggiornando il sistema con hardware supplementare in grado di garantire un incremento nel livello di sicurezza. È quest'ultimo l'ambiente all'interno del quale possono trovare luogo soluzioni commerciali come Entropy Key, NeuG, QUANTIS e qStream, oltre alla proposta discussa in questa tesi.

Discorso a parte va condotto per i sistemi embedded. Questi dispositivi concentrano in spazi ridotti hardware e software costruiti *ad hoc* per eseguire applicazioni specifiche. Ad esempio, la centralina di un'autovettura gestisce la propulsione del veicolo o un router wireless mantiene attiva la connessione alla rete. A differenza di sistemi informativi complessi, come PC e server, i quali richiedono capacità computazionali e di memorizzazione elevate, la prerogativa principale dei sistemi embedded è il basso costo.

Grazie all'evoluzione del concetto di Internet of Things [29], sempre più dispositivi elettronici hanno la necessità di connettersi in rete e di comunicare tra loro in maniera sicura e autenticata, andando a comporre una struttura pervasiva, avente un numero enorme di singoli *device*. Perché tale evoluzione sia economicamente ed energeticamente sostenibile, ognuno di questi *device* deve rispettare rigidi criteri, quali:

- basso costo
- basso consumo energetico
- ingombro ridotto
- sicurezza nelle comunicazioni

Confrontando le soluzioni evidenziate sopra, risulta evidente che, almeno per ora, le soluzioni integrate di Intel® e VIA® non onorino al meglio i primi tre punti. Al contrario, la nostra proposta erige le proprie fondamenta sui criteri elencati sopra, dimostrandosi un ottimo compromesso nei casi di retrofit di sistemi informativi datati ed embedded system privi di hardware dedicato allo scopo.

Produttore / Modello	Fenomeno	Prezzo [$]	Throughput
Simtec Electronics / Entropy Key	A valanga	43	32 Kbit/s [9]
TectroLabs / TL100	A valanga	329	1.4 Mbit/s
ID Quantique SA / Quantis-USB	Fotonico	1350	4 Mbit/s [52]
ID Quantique SA / Quantis-PCI-4	Fotonico	3000	16 Mbit/s [52]
Comscire / PQ4000KU	Shot	895	4 Mbit/s
Comscire / PQ32MU	Shot	1495	32 Mbit/s
LETech / GRANG-Mini	Termico	NA	16 Mbit/s
LETech / GRANG-EMB-2CH	Termico	NA	56 Mbit/s
LETech / GRANG-PCIC-8CH	Termico	NA	400 Mbit/s
LETech / GRANG-SATA	Termico	NA	400 Mbit/s
LETech / GRANG-Server	Termico	NA	1200 Mbit/s
LETech / GRANG-24CH	Termico	NA	4400 Mbit/s
Flying Stone Technology / FST-01	Quantizzazione	37	560 Kbit/s
TRNG98 / TRNG9880	Termico	37	50 Kbit/s
TRNG98 / TRNG9803	Termico	150	72 Kbit/s
TRNG98 / TRNG9815	Termico	860	550 Kbit/s
Araneus / Araneus Alea I	A valanga	NA	100 Kbit/s [41]
Quintessence Lab / qStream	Fotonico	NA	2 Gbit/s [44]

Tabella 3.2: Tabella di confronto tra TRNG in commercio.

3.4 Confronto tra TRNG in commercio

Per conferire all'elaborato un punto di vista in stretto legame con il mondo elettronico industriale, abbiamo ritenuto importante anche l'aspetto economico. Per tale motivo, se da un lato è bene evidenziare le prestazioni dei TRNG, dall'altro lato va tenuto in considerazione il prezzo di vendita. In Tab. 3.2 sono riportati svariati TRNG disponibili sul mercato, per ognuno dei quali si riportano le informazioni relative alla sorgente di rumore utilizzato, prezzo di vendita (quando disponibile) e throughput massimo raggiungibile (quello dichiarato dal produttore). La tabella non vuole essere esaustiva, ma fornire solo parametri di comparazione per valutare la proposta discussa in questa tesi.

Capitolo 4

Dinamiche caotiche

Come noto in letteratura [8] [2], sistemi dinamici che integrano elementi non lineari possono esibire comportamenti singolari, incluse traiettorie irregolari aperiodiche molto simili a rumore, oltre a un'estrema sensibilità alle condizioni iniziali. Questa forte dipendenza può portare due sistemi identici, con condizioni iniziali apparentemente uguali, a terminare in due stati completamente differenti. Generalmente, comportamenti di questo tipo vengono detti caotici e, come è facile intuire, ben si prestano alla generazione di numeri casuali [33] [55].

Sebbene le traiettorie prodotte da questi sistemi siano simili a rumore, l'uscita di un sistema caotico generalmente ha una distribuzione non uniforme e mostra correlazioni (dovute alla natura deterministica del processo di generazione). Per questo motivo è spesso necessario un processo di digitalizzazione, decorrellazione e bilanciamento. È bene evidenziare questa situazione di frontiera tra mondo deterministico e mondo stocastico sulla quale ci troviamo ad operare. Infatti, comportamenti caotici possono essere descritti accuratamente in termini stocastici, a fronte di un modello che è perfettamente deterministico. Tale posizione di confine è vantaggiosa per un TRNG, in quanto permette di attingere proprietà da entrambi i mondi: impredicibilità da quello stocastico e modelli matematici direttamente traducibili in architetture di *analog processing* da quello deterministico. Quindi, la natura dei sistemi caotici che andremo a studiare avrà sempre un modello deterministico.

Al fine di implementare un sistema caotico, è utile cercare di ottenere modelli deterministici dello stesso, in quanto l'architettura sarà semplicemente una tradu-

zione degli operatori che compaiono nel modello in componenti elettronici. La differenza, rispetto ai PRNG, è data dall'assenza di quantizzazione nelle variabili di stato del sistema, le quali però possono essere elaborate esclusivamente su hardware analogico.

Come già evidenziato nel Cap. 3, le dinamiche caotiche possono essere considerate un sistema di fruizione per fenomeni fisici rumorosi, in quanto, grazie alla proprietà di sensibilità alle condizioni iniziali, qualunque incertezza, manifestata sullo stato iniziale o introdotta nei vari stadi di elaborazione, viene amplificata dal sistema caotico. Il metodo operativo che stiamo ora esaminando nel dettaglio è un meccanismo di fruizione alla pari di quelli già considerati in precedenza nella Sez.3.1.4.

Ad oggi, si ritiene che molti fenomeni fisici rumorosi, considerati impredicibili, risultino tali in virtù delle dinamiche microcosmiche caotiche presenti al loro interno. In tali condizioni, è chiaro che piuttosto che impiegare fenomeni naturali, può essere conveniente sintetizzare sistemi artificiali che siano essi stessi sede di dinamiche caotiche. Infatti, attingendo in ultima istanza allo stesso tipo di fenomeno, nel primo caso occorre adattarsi a quanto i fenomeni naturali mettono a disposizione. Ciò è invariabilmente troppo complesso per utilizzare metodi formali di analisi. Nel secondo caso, invece, è possibile restringersi a quelle dinamiche caotiche per le quali alcuni strumenti di analisi sono disponibili, con evidenti vantaggi.

Esistono svariate tipologie di dinamiche caotiche, esaminate nel corso degli anni. Prime fra tutte, in ordine temporale, quelle tempo-continue. Tuttavia, risulta conveniente, allo scopo di semplificare la successiva analisi, restringere il campo ai soli sistemi unidimensionali tempo-discreti per i quali si hanno già validi strumenti matematici. Anzi, fra tutte le dinamiche caotiche appartenenti a tale classe, selezioniamo quelle più semplici da analizzare, ovvero le Piece-Wise Affine Markov Maps (PWAM) [2] [12]. Questa specifica classe di modelli non-lineari, come già detto, fa riferimento a sistemi caotici unidimensionali tempo-discreti, nei quali la variabile di sistema x è aggiornata in

$$x_{n+1} = M(x_n) \tag{4.1}$$

dove $M : [-1, 1] \to [-1, 1]$:

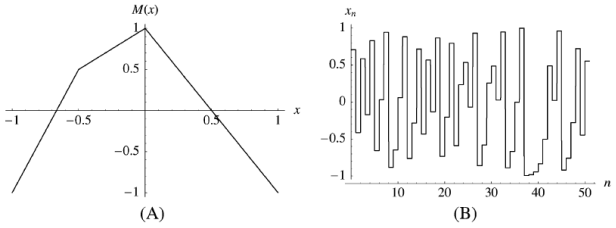


Figura 4.1: Esempio di mappa caotica (A); tipica traiettoria in uscita (B). (fonte [50])

1. è *non singolare*, nel senso che intervalli di misura non nulli non possono essere mappati da M su intervalli di misura nulli;

2. è *esatta*, nel senso che qualsiasi intervallo di misura non nullo viene eventualmente ampliato da M in un intervallo la cui misura è la stessa di $[-1, 1]$;

3. è tale che una partizione dell'intervallo $X_i,\ i = 0, ..., p-1$ di $[-1, 1]$ esiste affinché:

   (a) M è affine su ogni X_i;

   (b) i punti di ogni partizione sono mappati sui punti di partizione.

dove M prende il nome di mappa ed è scelta in modo tale che il sistema esibisca per l'appunto le caratteristiche di una dinamica caotica ed in prima istanza, sensibilità alle condizioni iniziali ed irregolarità delle traiettorie. Soddisfatte queste condizioni, dalla teoria matematica otteniamo strumenti di analisi molto potenti, grazie ai quali possiamo verificare e studiare i sistemi caotici. In particolare, saremo in grado di predire con precisione quale sarà la statistica delle sequenze generate in uscita, sia in termini di funzione densità di probabilità (PDF) che in termini di correlazione. Tali previsioni sono possibili in quanto i sistemi caotici, osservati attraverso un'opportuna funzione di uscita, sono modellabili come catene di Markov [12].

Imponendo una condizione iniziale x_0 e iterando la mappa M, verranno generate traiettorie irregolari che spaziano nell'intervallo $[-1, 1]$. Un esempio è riportato

in Fig. 4.1, nel quale è visibile in (A) la mappa

$$M(x) = \begin{cases} 3x + 2, & \text{for } x < -1/2 \\ x + 1, & \text{for } -1/2 \leq x < 0 \\ -2x + 1, & \text{altrove} \end{cases} \tag{4.2}$$

e in (B) la tipica traiettoria prodotta. Se M è caotica, anche cambiando x_0 di una piccolissima quantità, porta ad ottenere traiettorie completamente differenti, che, per n crescente, progressivamente (ed esponenzialmente) si allontanano rispetto al caso di riferimento. Quindi, anche in assenza di rumore, il comportamento del sistema è intrinsecamente impredicibile, in quanto la limitata precisione con la quale conosciamo lo stato del sistema ad ogni istante previene qualsiasi previsione a lungo termine.

Sistemi caotici basati su PWAM hanno natura sia deterministica che stocastica ed è quindi necessario adottare metodologie statistiche per la loro analisi [12]. Intuitivamente, grazie all'osservazione contemporanea di un numero elevato di traiettorie, generate da altrettante differenti condizioni iniziali, è possibile estrapolare comportamenti tipici e proprietà generali di un dato sistema caotico. Supponiamo di generare un numero enorme di condizioni iniziali x_0, aventi PDF $\rho_0(x)$. Applicando M, si ottiene un uguale numero di valori x_1. Questi sono distribuiti all'interno dell'intervallo $[-1, 1]$ secondo la PDF $\rho_1(x)$, la quale può essere ricavata da $\rho_0(x)$ conoscendo M. Ogni nuova iterazione produrrà una differente PDF, che possiamo legare alla precedente attraverso un operatore funzionale (dipendente da M), chiamato operatore di Perron-Frobenius e indicato con P_M [2] [12].

Mentre la mappa M trasforma punti in punti, l'operatore P_M ha il compito di trasformare PDF in PDF. Da notare come, nonostante M sia non-lineare, P_M è lineare. Inoltre, se M soddisfa le proprietà elencate sopra, P_M è costrittiva, ovvero, asintoticamente, esiste una sola PDF ρ_{inf}, qualsiasi sia la funzione densità di probabilità di partenza ρ_0 (a condizione che ρ_0 sia scelta con cura [12]). L'esistenza di una densità di probabilità invariante è estremamente importante. Infatti, le proprietà 1 e 2 sopracitate implicano l'ergodicità del sistema, da cui deriva la possibilità di scambiare medie statistiche con medie temporali. In altri termini, lo stesso comportamento che si osserva studiando un grandissimo numero di sistemi

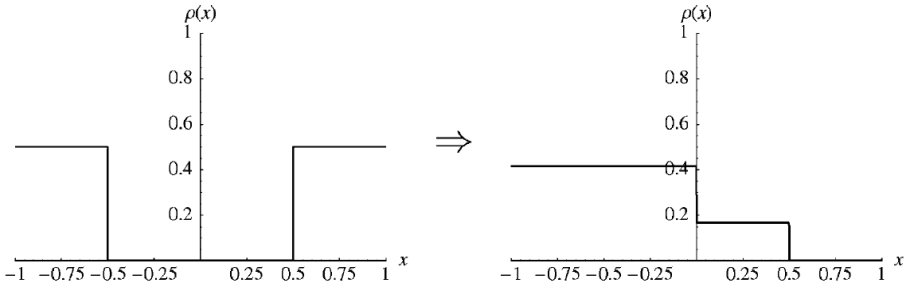


Figura 4.2: Esempio di trasformazione compiuto dalla mappa in Fig. 4.1 sulla PDF a gradino. (fonte [50])

inizializzati con diverse condizioni iniziali, può venire osservato su un solo sistema, a patto di seguine il comportamento per un tempo sufficientemente lungo. Questo significa che i valori di ogni traiettoria tipica del sistema devono distribuirsi secondo la densità invariante. In altre parole, trovando la densità invariante si ottiene anche la statistica del prim'ordine di qualunque traiettoria tipica. Essendo il modello deterministico, da quest'ultima è poi possibile estrarre qualunque momento di ordine superiore. Quindi l'operatore di Perron-Frobenius e l'ottenimento della densità invariante per l'operatore consentono, in ultima istanza, di risalire all'intera statistica delle traiettorie del sistema caotico.

A questo punto, è interessante osservare cosa succede se, invece di considerare qualsiasi possibile densità iniziale ρ_0, ci limitiamo a considerare PDF con andamenti a gradino su X_i: in questo caso, proprio grazie alla possibilità di definire una partizione su cui tutti i tratti della mappa sono affini, tutte le densità $\rho_1, ..., \rho_{\text{inf}}$ diventano a gradino. La scoperta appena fatta è importante, in quanto permette di considerare le probabilità finite della variabile di stato x di trovarsi in uno degli intervalli X_i, invece di dover tenere conto dell'intera PDF. In altri termini, ad ogni passo n, possiamo andare a sostituire ρ_n con il vettore di probabilità $\mathbb{P}_n = (P_{n,0}, ..., P_{n,p-1})$. Allo stesso modo, l'operatore finito dimensionale K_M può sostituire l'operatore funzionale P_M. Essendo K_M lineare e finito dimensionale può essere rappresentato in forma di matrice, sotto il nome di *keading matrix*, grazie alla quale possiamo scrivere che $\mathbb{P}_{n+1} = \mathbb{P}_n \, K_M$. Aspetto interessante è che anche l'operatore basato sulle probabilità discrete è compressivo e tale da avere convergenza su un vettore di probabilità invariante. Questo significa che la densità di probabilità invariante per la mappa è proprio costante a tratti e può

essere ricavata utilizzando l'operatore K_M.

Gli elementi $k_{i,j}$ di K_M rappresentano la probabilità che una traiettoria passante in X_i all'instante n cada in X_j all'istante $n+1$. Sotto queste premesse, K_M è la descrizione di una *macchina a stati probabilistica* o di una *catena di Markov*, incorporate nelle dinamiche caotiche [12]. Più precisamente, tale macchina è nello stato $\mathbf{x}_i$ quando lo stato del sistema caotico si trova nell'intervallo X_i. Le probabilità di transizione tra stati sono rappresentati dalla kneading matrix.

Alcune catene di Markov sono in grado, data la loro costruzione, di generare simboli Indipendenti e Identicamente Distribuiti (IID). Altre, invece, non mostrano direttamente questa proprietà. Spesso queste ultime possono comunque essere ridotte al primo caso, vuoi in maniera esatta, vuoi in modo molto ben approssimato, da cui risulta oramai evidente la possibilità di generare variabili *true random* sfruttando una sorgente PWAM. Una schematizzazione del generatore appena descritto e della relativa astrazione matematica è visibile in Fig. 4.3. Nella figura è presente la catena di Markov ottenuta per analisi dell'operatore di Perron-Frobenius (al centro). Inoltre, è presente un'ulteriore catena ottenuta per semplice aggregazione degli stati della precedente (a destra). È evidente come quest'ultima è del tutto equivalente alla catena che corrisponde all'esperimento del lancio di una moneta (testa o croce), dimostrando quindi la capacità del sistema di generare eventi binari casuali, quando lo stato viene osservato attraverso un'opportuna funzione di quantizzazione. È interessante osservare la catena di Markov a cui si arriva, mediante una semplice aggregazione degli stati presenti nella catena implicata dall'operatore di Perron-Frobenius.

Per progettare un TRNG basato su una mappa PWAM è necessario seguire i passi illustrati in precedenza, ovvero identificare una catena di Markov adatta a generare bit IID, realizzandone (se necessario) una versione ridotta e ottenendo infine una sorgente caotica PWAM che la implementa.

4.1 Sorgenti di rumore basate su mappe PWA

Sorgenti caotiche necessitano di hardware analogico, il quale consente variazioni arbitrariamente piccole dello stato del sistema, condizione essenziale per rivelare la *sensibilità alle condizioni inziali*. Su hardware digitale questa libertà sarebbe

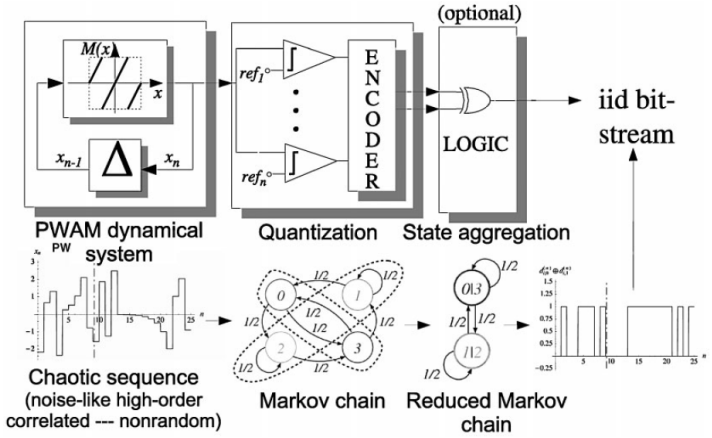


Figura 4.3: Schematizzazione delle fasi di signal processing presenti in un TRNG basato sul *chaos* che sfrutti mappe PWAM e quantizzazione dell'uscita. Al di sotto è visibile la relativa astrazione matematica. (fonte [51])

compromessa, vista la natura numerica dei circuiti digitali. Inoltre, si violerebbe una delle condizioni per poter applicare la teoria vista sopra, in quanto le mappe computate digitalmente sarebbero singolari (vedi proprietà 1 nella sezione precedente). Quindi, il metodo esposto precedentemente non è sufficiente all'implementazione di TRNG su dispositivi digitali programmabili, come le Field Programmable Gate Array (FPGA). Al contrario, la controparte analogica delle FPGA, ovvero le Field Programmable Analog Array (FPAA), possono trarre vantaggio del metodo per la generazione di numeri casuali descritto in precedenza [51].

Integrate in grande numero e agevolmente programmabili sulle FPAA, le primitive analogiche indispensabili per ottenere dinamiche complesse, sono solo alcune:

- blocchi di soglia o comparazione per la funzione di quantizzazione;
- blocchi di soglia, comparazione o rettificazione connessi ad operatori di combinazione lineare per la sintesi di mappe PWA;
- blocchi di memorizzazione per mantenere lo stato corrente del sistema.

Fortunatamente, i sistemi PWAM con osservabili quantizzati possono essere progettati sfruttando solamente questo ristretto insieme di primitive lineari e non-lineari.

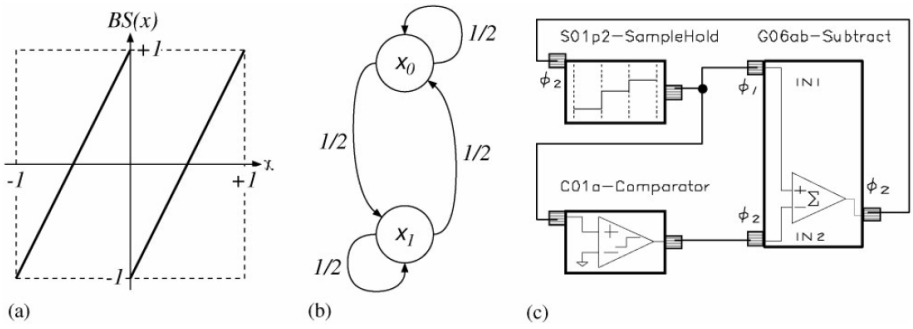


Figura 4.4: Sorgente caotica PWAM basata su Bernoulli shift (a); catena di Markov incorporata (b); implementazione FPAA (c). (fonte [51])

4.1.1 Esempio di mappa PWA: Bernoulli Shift

Come primo esempio, descriviamo l'implementazione di un TRNG basato su Bernoulli shift. La mappa implementata, visibile in Fig. 4.4(a), è

$$M_{BS}(x) = 2x \bmod(2) - 1 \tag{4.3}$$

Gli intervalli sui quali la mappa è PWAM sono chiaramente $\{X_0 = [-1, 0[,\ X_1 = [0, 1]\}$, dai quali si ottiene la matrice delle probabilità di transizione

$$K_{BS} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{4.4}$$

La catena di Markov incorporata è visibile in Fig. 4.4(b). Quella rappresentata è la già nota catena 'testa o croce' mostrata precedentemente in Fig. 4.3, la quale è in grado di produrre sequenze di simboli IID senza un ulteriore post-processing. Per poter ricavare tale catena di Markov è stato sufficiente quantizzare lo stato del sistema mediante una funzione di comparazione, come

$$y_n = f(x_n) = \mathrm{cmp}(x, 0) \tag{4.5}$$

$$\mathrm{cmp}(a, b) = \begin{cases} 1, & \text{se } a > b \\ -1, & \text{altrove} \end{cases} \tag{4.6}$$

In Fig. 4.4(c) è mostrata l'implementazione della catena, nella quale le pri-

mitive impiegate sono un blocco di comparazione, uno di differenza e uno di memorizzazione (Sample and Hold). Da notare come $M_{BS}(x)$ sia stata riformulata in $2x - f(x)$.

Test effettuati su un prototipo implementato su FPAA hanno mostrato un throughput sperimentale di $5\,\mathrm{Mbit/s}$, sfruttando un clock di $1\,\mathrm{MHz}$ [51]. Lo stream binario prodotto in uscita è comparabile a rumore bianco, sebbene sia necessario applicare alla mappa una minima distorsione (nell'ordine dell'1%) per ottenere un comportamento affidabile.

4.1.2 Selezione della mappa PWA

Per implementare un sistema caotico su PWAM è richiesta una scelta accurata della mappa, in quanto alcune di esse possono esibire problemi di robustezza. Tali condizioni si verificano quando i punti della mappa vengono a coincidere con i punti limite dello spazio degli stati. Questo consentirebbe allo stato di evadere dallo spazio degli stati, portando il sistema ad assumere stati che non appartengono alla mappa. In particolare la mappa Bernoulli shift (come anche la Ten map e altre) ha problemi di robustezza, in quanto i suoi estremi coincidono sono i punti limite superiore ed inferiore, individuati dall'intersezione tra la mappa e la retta bisettrice principale dello spazio degli stati.

Inoltre, è evidente che l'implementazione della mappa deve essere analogica e possedere livelli di accuratezza elevati. Purtroppo, tali caratteristiche sono complesse da ottenere, in quanto la progettazione e realizzazione di circuiti analogici molto accurati è costosa, sia in termini di risorse economiche che in termini di tempo. Tuttavia, esiste una classe di circuiti analogici perfetta per il nostro scopo: i convertitori ADC. Infatti, gli ADC sono i componenti elettronici analogici che hanno visto i maggiori e migliori investimenti, allo scopo di renderli sempre più precisi, veloci ed economici. Quindi, non deve sorprendere la scelta di impiegare mappe ADC per l'implementazione del nostro sistema caotico.

4.1.3 Proposta implementativa: ADC Map

Viste le considerazioni precedenti, proponiamo ora un'architettura incentrata su ADC, sulla quale si baserà il TRNG oggetto di questa proposta di tesi (vedi 5).

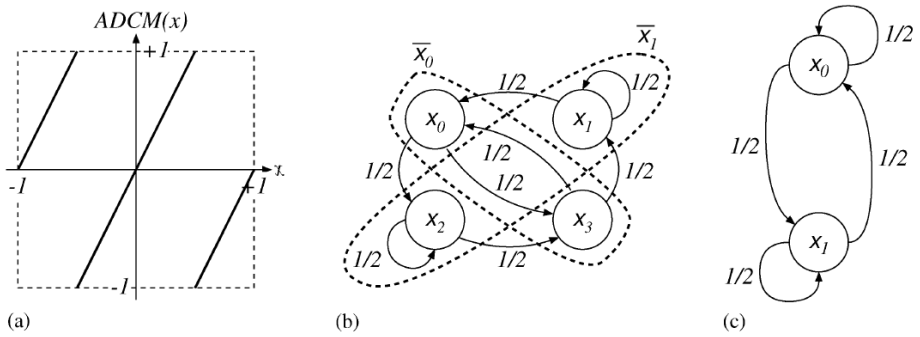


Figura 4.5: Sorgente caotica PWAM basata su ADC a pipeline (a); catena di Markov incorporata (b); e catena ridotta (c). (fonte [51])

Nel caso specifico, consideriamo un generatore caotico TRNG progettato sulla più semplice incarnazione di un convertitore ADC a *pipeline* a $1+\frac{1}{2}$ bit per stadio [49]. La mappa realizzata, visibile in Fig. 4.5(a), è

$$M_{ADC}(x) = (2x-1)\mathrm{mod}(2) - 1 \tag{4.7}$$

In questo caso gli intervalli sui quali la mappa è PWAM sono $\{X_0 = [-1, -\frac{1}{2}[$, $X_1 = [-\frac{1}{2}, 0[$, $X_2 = [0, \frac{1}{2}[$, $X_3 = [\frac{1}{2}, 1]\}$, dai quali si ottiene la matrice delle probabilità di transizione

$$K_{ADCM} = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{pmatrix} \tag{4.8}$$

Come mostrato in Fig. 4.5, è evidente che la mappa M_{ADC} non esibisce problemi di robustezza. Infatti, non esistono punti della mappa che coincidono con punti limite dello spazio degli stati. Questo consente di ottenere il comportamento desiderato: lo stato del sistema caotico rimane confinato all'interno dello spazio degli stati, anche in presenza di perturbazioni esterne.

La catena di Markov incorporata è visibile in Fig. 4.5(b) (ignorando le righe tratteggiate). La catena rappresentata ha quattro stati e, a causa della sua attuale struttura, è incapace di generare simboli IID. Per verificare questa affermazione, consideriamo, per esempio, di chiamare $\mathbf{x}^{(n)}$ il generico stato al tempo (n) e

valutiamo la probabilità condizionata $\Pr(\mathbf{x}^{(n)} = \mathbf{x}_1 | \mathbf{x}^{(n-1)} = \mathbf{x}_0)$. Ovviamente, tale probabilità è differente se consideriamo il caso $\Pr(\mathbf{x}^{(n)} = \mathbf{x}_1 | \mathbf{x}^{(n-1)} = \mathbf{x}_3)$. Quindi, la probabilità di trovarsi in ogni istante di tempo nello stato $\mathbf{x}_1$ non è indipendente dallo stato precedente.

Come già anticipato, le catene di Markov possono essere ridotte. Fortunatamente, in questo caso particolare, è sufficiente aggregare gli stati a due a due, evidenziando i due nuovi stati $\bar{\mathbf{x}}_0$ e $\bar{\mathbf{x}}_1$ [7] [49]. Seguendo le linee tratteggiate in Fig. 4.5(b), supponiamo che il nuovo sistema si trovi nello stato $\bar{\mathbf{x}}_0$ quanto il sistema originale è nello stato $\mathbf{x}_0$ o $\mathbf{x}_3$ e nello stato $\bar{\mathbf{x}}_1$ altrimenti. Il nuovo sistema è descritto dalla catena in Fig. 4.5(c), la quale coincide nuovamente con la catena 'testa o croce'. Come già detto in precedenza, questo tipo di catena può produrre simboli IID. Quindi, la mappa M_{ADC} è adatta alla sintesi di TRNG.

Test effettuati su un prototipo implementato su FPAA hanno mostrato un throughput sperimentale di $2\,\mathrm{Mbit/s}$, sfruttando un clock di $1\,\mathrm{MHz}$ [51]. Sebbene sia presente una notevole riduzione del data rate rispetto alla mappa M_{BS}, è necessario tenere in conto che la mappa M_{ADC} ha il grande vantaggio di essere intrinsecamente robusta [36] [49]. In altri termini, non è necessario applicare alcun tipo di distorsione alla mappa per assicurare un comportamento affidabile, a differenza del caso precedente.

4.2 Implementazione di un TRNG basato su ADCM

Come già anticipato nella Sez. 4.1.3, architetture incentrate su ADC possono essere sfruttate in modo conveniente per generare simboli casuali IID. Questo è possibile in quanto gli ADC implementano implicitamente una funzione PWAM ogni volta che si prende in considerazione l'errore di conversione o residuo. Per poter derivare quest'ultimo è necessario conoscere la funzione di quantizzazione $Q(x)$ dell'ADC, in modo da ricavare

$$E(x) = x - Q(x) \tag{4.9}$$

dove x è ovviamente il segnale in ingresso. Considerando un ADC con una risoluzione di k bit, otteniamo che $Q(x)$ è una funzione crescente avente 2^k gradini.

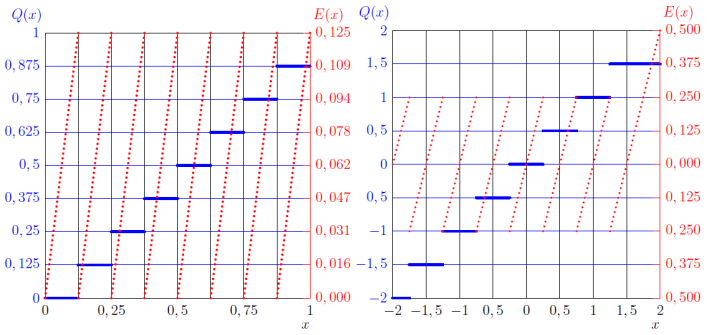


Figura 4.6: Esempi di funzioni di quantizzazione e di errore. A sinistra, un ADC operante sull'intervallo $[0, 1]$ in ingresso, in assenza di arrotondamento (viene sempre scelto il livello inferiore). A destra, un ADC operante sull'intervallo $[-2, 2]$ in ingresso con arrotondamento (al livello di quantizzazione più vicino). In entrambi i casi la risoluzione è 3 bit. (fonte [48])

Conseguentemente, $E(x)$ è una funzione a 2^k rampe. Sia $Q(x)$ (linea continua) che $E(x)$ (linea tratteggiata) sono visibili in Fig. 4.6, al variare dell'ingresso x in ascissa. In figura sono rappresentati (riportati come esempio) due grafici distinti, in quanto la funzione di quantizzazione dipende dal tipo di ADC, unipolare o bipolare, con arrotondamento (al livello di quantizzazione più vicino) o senza arrotondamento (optando, per esempio, sempre per il livello di quantizzazione minore).

Generalmente si hanno informazioni limitate sulle funzioni $Q(x)$ e $E(x)$, in quanto il circuito ADC rende disponibile in uscita il solo vettore binario $\mathbb{B}$ determinato attraverso la funzione $N : [R_{min}, R_{max}] \mapsto \{0, 1\}^k$ dove R_{min} insieme a R_{max} identifica l'intervallo in ingresso e N combina le funzioni di quantizzazione e codifica. In questi casi, per poter ottenere $Q(x)$ e in seguito $E(x)$, è sufficiente introdurre nell'architettura un blocco complementare all'ADC, ovvero un Digital to Analog Converter (DAC), seguito da un blocco di somma, come mostrato in Fig. 4.7 [48].

4.2.1 Parametri liberi dell'architettura

Per poter approfittare della teoria affrontata nel Cap.4, la mappa implementata dall'architettura deve rispondere alle proprietà 1, 2 e 3. In particolare, l'intervallo $I \subseteq [R_{min}, R_{max}]$, chiamato anche Invariant Set (IS), deve essere mappa-

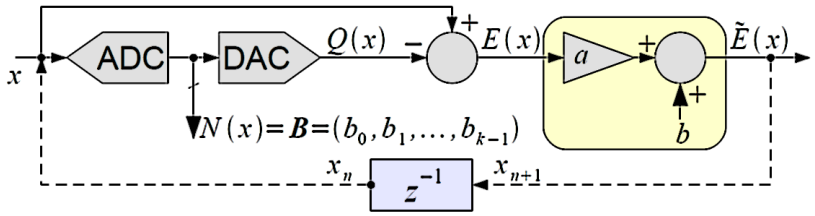


Figura 4.7: Estensione di un'architettura ADC per ottenere una rappresentazione fisica di $Q(x)$, $E(x)$ e $\tilde{E}(x)$, atta a implementare un sistema dinamico autonomo 1-D. (fonte [48])

R_{min}, R_{max}	Arrotondamento	a	b
$0, R$	No	$1 < a < 2^k$	$0 < b < R - aq$
$0, R$	Si	$1 < a < 2^k$	$a\frac{q}{2} < b < R - (a+1)\frac{q}{2}$
$-\frac{R}{2}, \frac{R}{2}$	No	$1 < a < 2^k$	$-\frac{R}{2} < b < \frac{R}{2} - aq$
$-\frac{R}{2}, \frac{R}{2}$	Si	$1 < a < 2^k$	$-\frac{R}{2} + a\frac{q}{2} < b < \frac{R}{2} - (a-1)\frac{q}{2}$

Tabella 4.1: Criteri di selezione dei parametri a e b a seconda del tipo di ADC e dalla presenza o assenza di arrotondamento. In questo caso, il passo di quantizzazione q vale $R/2^k$.

to esattamente su se stesso. Per rispettare questa condizione, viene introdotta, mediante l'applicazione di una trasformazione affine idonea, la nuova funzione $\tilde{E}(x) = aE(x) + b$, nella quale la combinazione $a \neq 1$ e $b \neq 0$ si verifica quando, in funzione di E, il dominio dell'IS e la sua immagine non coincidono. Il sistema tempo discreto autonomo monodimensionale così costruito è visibile in Fig. 4.7, dove la variabile di sistema x è aggiornata in

$$\begin{cases} x_{n+1} = \tilde{E}(x_n) \\ \mathbb{B}_n = N(x_n) \end{cases} \tag{4.10}$$

Allo scopo di rispettare le condizioni appena citate, i parametri a e b devono essere scelti con cura. In particolare, a dovrebbe essere tale che $1 < a < 2^k$, mentre b dovrebbe essere tale da avere un IS *ragionevolmente largo* e *ben centrato* all'interno dell'intervallo disponibile $[R_{min}, R_{max}]$. Queste ultime considerazioni verranno meglio approfondite nella prossima sezione. Inoltre, b dipende dal

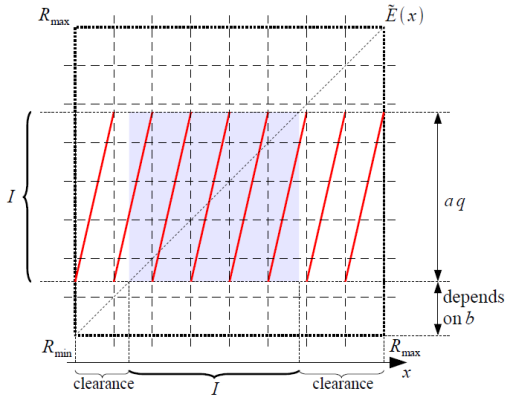


Figura 4.8: Ruolo dei parametri a e b nella determinazione dell'Invariant Set e margini necessari ad assicurare operazioni affidabili. (fonte [48])

tipo di ingresso dell'ADC (unipolare o bipolare) e dalla presenza o assenza di arrotondamento, come evidenziato in Tab. 4.1.

Da notare, come mostrato in Fig. 4.8, il margine lasciato tra l'IS e il confine dell'intervallo $[R_{min}, R_{max}]$. Quest'ultimo assicura un comportamento robusto dell'architettura, anche nel caso in cui sia sottoposta a rumore o non-idealità dell'ADC. Inoltre, la condizione su a garantisce una pendenza dei rami tale da verificare un comportamento di $\tilde{E}$ di tipo *stretch and fold*[8], indispensabile al fine di ottenere un sistema caotico [12] [2].

Configurati correttamente i parametri a e b, ricorsivamente, a ogni ciclo, il sistema fornisce in uscita il vettore binario $\mathbb{B}$, che tiene traccia della traiettoria caotica dello stato x del sistema. Sebbene possano essere presenti ancora deboli correlazioni nella sequenza $\{\mathbb{B}_n\}$ così prodotta, quest'ultima rimane una sorgente di rumore ideale per la realizzazione di un TRNG [50].

4.2.2 Scelta ottima dei parametri

Al fine di rendere agevole la codifica e la distillazione della sequenza casuale in uscita dal generatore, è essenziale attuare una scelta ottima sui parametri liberi dell'architettura. A questo scopo, è necessario analizzare la statistica di $\{\mathbb{B}_n\}$,

[8]Meccanismo non lineare che consente di ripiegare all'interno di uno spazio degli stati limitato (folding) delle traiettorie che corrispondono ad un comportamento divergente (stretching).

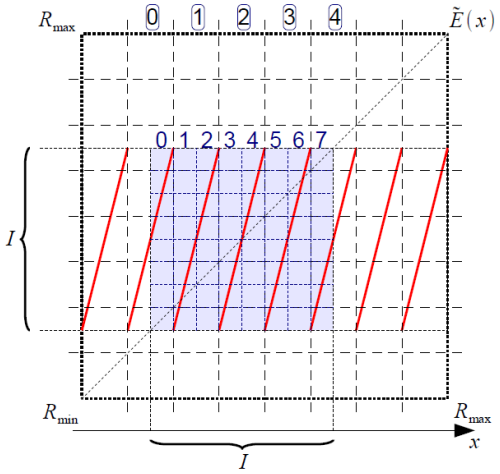
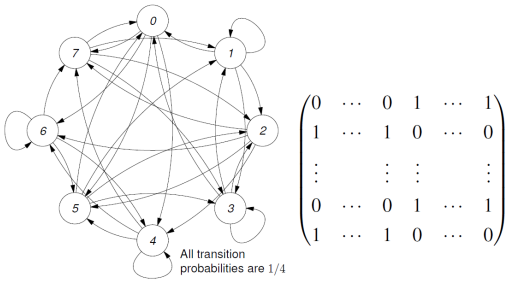


Figura 4.9: Esempio risultante dalla scelta ottima di a e b per un ADC a 3 bit con arrotondamento per difetto. (fonte [48])

nella quale sono celate le proprietà della funzione di trasformazione di stato $\tilde{E}$. Essendo $\tilde{E}$ una funzione PWAM, le probabilità di transizione tra stati della catena di Markov annessa possono essere ricavate attraverso l'osservazione diretta di $\tilde{E}$.

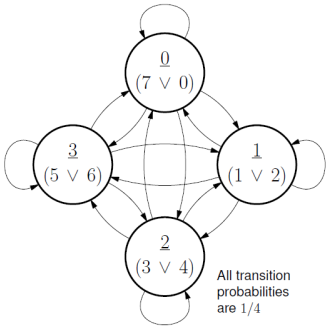
Per semplificare tale studio, è ragionevole scegliere a e b in modo tale che la partizione $\{X_i\}$ coincida con gli intervalli di quantizzazione dell'ADC [48]. Questo implica optare per $a = 2^{k-1}$, ovvero il più grande valore possibile tra le sole potenze del 2, in modo tale da amplificare il fenomeno dello *stretching* citato in precedenza. Apparentemente, una scelta ragionevole per b dovrebbe garantire un perfetto allineamento tra la partizione $\{X_i\}$ e i livelli di quantizzazione. Purtroppo, una decisione di questo tipo sarebbe alquanto sfortunata, una volta considerati possibili errori implementativi. Questi ultimi possono presentarsi facilmente sotto forma di offset, modificando l'insieme dei livelli di quantizzazione afferenti all'IS, portando ad inutili complicazioni nell'implementazione del distillatore. Quindi, per incrementare la robustezza del generatore si preferisce allineare i punti della partizione $\{X_i\}$ con il centro dei livelli di quantizzazione, come mostrato in Fig. 4.9. A tale scopo, è necessario imporre $b = mq$ (per ADC con arrotondamento) o $b = mq + q/2$ (per ADC senza arrotondamento) con $m \in \mathbb{Z}$, nel rispetto dei limiti definiti nella Tab. 4.1.

In Fig. 4.9 numeriamo, da 0 a $2^k - 1$, per x crescenti, gli intervalli di partizione $\{X_i\}$ (e quindi gli stati della catena di Markov). Fatto questo, possiamo costruire la



$$\begin{pmatrix} 0 & \cdots & 0 & 1 & \cdots & 1 \\ 1 & \cdots & 1 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 1 & \cdots & 1 \\ 1 & \cdots & 1 & 0 & \cdots & 0 \end{pmatrix}$$

Figura 4.10: Catena di Markov per $k = 3$ e matrice delle probabilità di transizione per k generico. (fonte [48])



$$\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$$

Figura 4.11: Catena di Markov aggregata per $k = 3$ e matrice delle probabilità di transizione per k generico (fonte [48])

catena di Markov risultante per $k = 3$ e la forma della matrice delle probabilità di transizione per k generico (vedi Fig. 4.10). Come già evidente dalla struttura della kneading matrix, una catena di Markov di questo tipo non può produrre simboli IID. Mostra comunque buone proprietà, dovute soprattutto alla regolarità della sua struttura, grazie alle quali si riesce ad ottenerne una versione ridotta.

Aggregando a due a due gli stati della catena in Fig. 4.10 si ottiene la catena ridotta visibile in Fig. 4.11, nella quale sono presenti 2^{k-1} stati aggregati (indicati con numeri sottolineati). Basandosi su questa nuova rappresentazione e sfruttando la teoria matematica in [48] [12], si può concludere che

- la catena aggregata descrive un processo di Markov che produce simboli IID (ovvero *true random*), coincidente alla descrizione della dinamica di un dado perfettamente bilanciato con 2^{k-1} facce;
- gli stati della catena aggregata possono essere messi in diretta relazione con

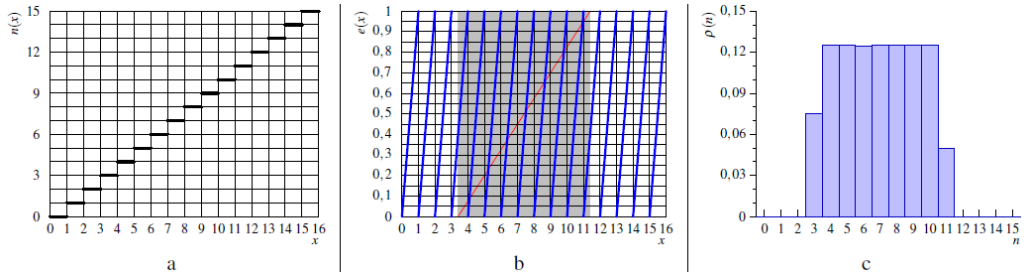


Figura 4.12: Caso ideale: funzione di quantizzazione per $k = 4$, $a = 2$ e $b = 3.4$ (a); funzione errore e IS del sistema caotico (area grigia) (b); istogramma di uscita (c). (fonte [47])

gli intervalli di quantizzazione, identificati in Fig. 4.9 con le etichette da $\boxed{0}$ a $\boxed{2^{k-1}-1}$.

Viste le considerazioni sopra, risulta evidente come una scelta ottima dei parametri a e b contraddistingue un'implementazione agevole del distillatore, ora in grado di codificare gli stati aggregati in soli $k-1$ bit (2^{k-1} codici binari).

4.2.3 Statistica della sorgente ADCM

Risulta ora evidente che, quando un ADC è portato in auto-oscillazione caotica, la statistica della sequenza digitale prodotta in uscita è completamente nota. Questo vale finché il sistema segue il proprio modello nominale. In caso contrario, sotto l'azione, ad esempio, di perturbazioni esterne, offset o non-idealità, la statistica dello stream prodotto in uscita si discosta dal comportamento atteso.

Beneficiando di quanto visto nella sezione precedente, possiamo dedurre informazioni sull'istogramma rappresentativo della probabilità associata ad ogni codice distillato in uscita dal generatore. In condizioni nominali, ognuno dei 2^{k-1} codici è *equiprobabile* con probabilità $1/2^{k-1}$, ad eccezione dei due più rami più esterni. Per questi, vale infatti la probabilità $1/2^k$. Quanto appena detto è evidente nell'istogramma visibile in Fig. 4.12, ottenuto mediante simulazione di un'architettura con ADC a $4\,\mathrm{bit}$ [47].

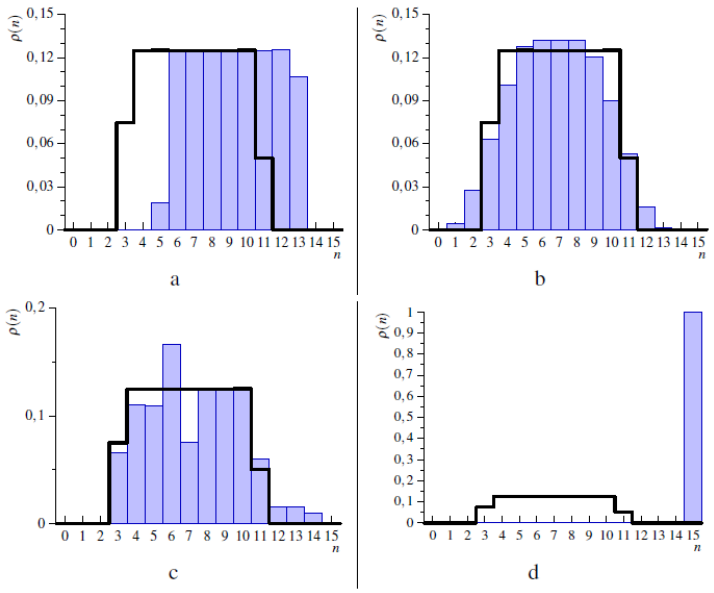


Figura 4.13: Caso non ideale: shift dell'istogramma a causa di un errore di offset (a); allargamento dell'istogramma dovuto a un errore di guadagno (b); problema di non-linearità su un singolo livello di quantizzazione (c); impossibilità di stabilire l'oscillazione caotica a causa di evidenti difetti nell'architettura. La linea più spessa mostra il comportamento atteso. (fonte [47])

Come confronto, in Fig. 4.13 sono riportati diversi esempi di istogrammi fuori specifica, i quali contraddistinguono sistemi caotici incapaci di generare simboli IID. Comportamenti anomali possono essere dovuti a

- presenza di *offset* sull'ingresso dell'ADC o alterazioni del parametro b rispetto alla specifica di progetto;
- *errore di guadagno* dell'ADC o alterazioni del parametro a rispetto alla specifica di progetto;
- *non-idealità* dell'ADC, imputabili a livelli di quantizzazione difformi.

L'errore dovuto all'offset produce una traslazione rigida dell'istogramma, facilmente correggibile mediante la regolazione del parametro b. Invece, l'errore di guadagno causa una dilatazione orizzontale dell'istogramma, il quale assume una distribuzione di tipo Gaussiano per errori di guadagno elevati. Un errore di questo tipo può essere corretto effettuando una calibrazione del parametro a. Infine,

non-idealità dell'ADC vengono riflesse direttamente sull'istogramma e difficilmente possono essere corrette, a meno di effettuare un post-processing mirato. Quindi, risulta evidente la necessità di un'accurata selezione della coppia ADC e DAC, siccome questi ultimi sono disposti in cascata nell'architettura del generatore (vedi Fig. 4.7). In particolare è fondamentale la scelta dell'ADC, il quale dovrebbe possedere Integral Non-Linearity (INL) ridotto [47].

4.2.4 Impredicibilità della sorgente ADCM

Ottenuta la dimostrazione dell'idealità della statistica prodotta in uscita dal TRNG basato sulla mappa M_{ADC} , è ora possibile mostrare come, una sorgente così costruita, garantisca completa impredicibilità. Fondamentalmente, è impossibile conoscere con accuratezza infinita lo stato del sistema, perché inevitabilmente affetto da rumore. Pertanto, data la sensibilità alle condizioni iniziali del sistema caotico celato nell'architettura, la probabilità di ottenere due traiettorie identiche in corse differenti è infinitesima.

Ipotizziamo una variabile di sistema rappresentata da una tensione, avente massimo swing V_{swing}. Inoltre, lo stato del sistema può essere misurato con un errore minimo di tipo Gaussiano con varianza σ_E^2. Detto questo, è impossibile identificare la condizione iniziale con una precisione relativa migliore di $\pm 3\sigma_E / V_{swing}$. In aggiunta è presente anche rumore di processo (Gaussiano) con varianza σ_P^2. Grazie alla pendenza dei rami della mappa M_{ADC} (vedi Fig. 4.5(a)), uguale a 2, la deviazione standard della tensione che rappresenta lo stato futuro x_1 del sistema diventa

$$\sigma_1 = \sqrt{(2\sigma_E)^2 + \sigma_P^2} \tag{4.11}$$

dove, nuovamente, x_1 non può essere determinato con una precisione relativa migliore di $\pm 3\sigma_1 / V_{swing}$. Ad ogni nuova iterazione del sistema aumenta l'incertezza sullo stato del sistema e al passo n otteniamo la varianza

$$\sigma_n = \sqrt{4^n \, \sigma_E^2 + \frac{1}{3}(4^n - 1)\sigma_P^2} \tag{4.12}$$

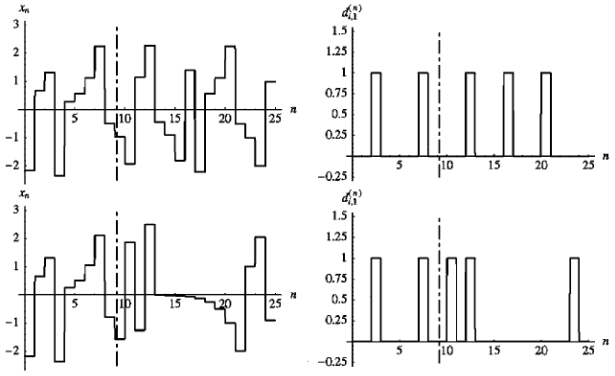


Figura 4.14: Nella colonna di sinistra, confronto tra due traiettorie prodotte dal medesimo generatore in condizioni iniziali uguali, completamente scorrelate dopo soli $9 \div 10$ passi; a destra, i simboli prodotti in uscita. (fonte [50])

Quando $6\sigma_n$ diventa comparabile con V_{swing}, anche conoscendo x_0, diviene impossibile stimare il valore di x_n o, ancora peggio, l'uscita digitale del sistema. Approssimativamente, dalla 4.12, questo avviene dopo $\tilde{n}$ passi, dove

$$\tilde{n} \approx \log_2 \left(\frac{V_{swing}}{6\sqrt{\sigma_E^2 + \frac{1}{3}\sigma_P^2}} \right) \tag{4.13}$$

Sebbene il valore di $\tilde{n}$ nella 4.13 sia solo un'approssimazione, questo fornisce un utile parametro di verifica dell'impredicibilità del sistema [12]. Infatti, minore è il numero di passi $\tilde{n}$, maggiore è la velocità con cui decadono le correlazioni tra due possibili traiettorie aventi condizioni iniziali pressoché identiche. Le dirette conseguenze di questa affermazione sono due.

Come prima conseguenza, se cerchiamo di effettuare una seconda corsa sul medesimo generatore, inizializzandolo ad una condizione iniziale nota (la quale potrebbe essere stata intercettata in un precedente attacco), dopo circa $\tilde{n}$ passi il flusso di bit casuali prodotti dal generatore diventa IID. Per illustrare questo fenomeno, in Fig. 4.14 sono mostrate due possibili corse del medesimo generatore, inizializzato allo stesso modo. Se consideriamo uno swing di $5\,\mathrm{V}$ e un errore σ_n pari a $1\,\mathrm{mV}$, dalla 4.13 otteniamo $\tilde{n} \approx 9 \div 10$. È evidente, confrontando i due grafici, che lo stato del sistema e conseguentemente il valore binario prodotto, differiscano dopo sole 9-10 iterazioni.

Come seconda osservazione, notare come, per ottenere un comportamento completamente impredicibile, sia necessario scartare i bit generati dalle sole prime $\tilde{n}$ iterazioni del sistema. Quindi, l'architettura proposta rappresenta un modo agile e relativamente semplice per produrre sequenze con proprietà di impredicibilità e irripetibilità.

Capitolo 5

Proposta di TRNG a basso costo

Proponiamo di seguito la nostra implementazione di un True Random Number Generator basato su dinamica caotica PWAM. In particolare, sfrutteremo i concetti esposti nel Cap. 4, le proprietà della mappa M_{ADC} e l'architettura mostrata in Fig. 4.7.

5.1 Architettura del sistema

Nel Cap. 4 abbiamo teorizzato la generazione di bit IID iterando la mappa PWA M_{ADC} . In [51] si è dimostrato sperimentalmente come sia possibile realizzare tale sistema su FPAA. Ora, puntiamo a raggiungere lo stesso traguardo, ottenuto su FPAA, impiegando solamente elettronica discreta *low cost*. Gli elementi fondamentali dell'architettura del sistema per la creazione della mappa M_{ADC} sono visibili in Fig. 5.1.

Partendo dal convertitore AD, fulcro del nostro sistema, si aggiunge in cascata la sua funzione complementare, il convertitore DA. Un bus digitale binario trasfe-

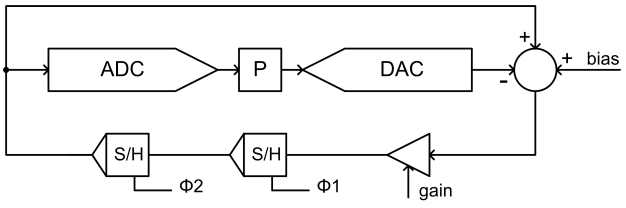


Figura 5.1: Schema a blocchi del generatore TRNG basato su dinamica caotica.

R_{min}, R_{max}	Arrotondamento	a	b'
$0, R$	No	$1 < a < 2^k$	$0 < b' < \frac{R}{a} - q$
$0, R$	Si	$1 < a < 2^k$	$\frac{q}{2} < b' < \frac{R}{a} - (1 + \frac{1}{a})\frac{q}{2}$
$-\frac{R}{2}, \frac{R}{2}$	No	$1 < a < 2^k$	$-\frac{R}{2a} < b' < \frac{R}{2a} - q$
$-\frac{R}{2}, \frac{R}{2}$	Si	$1 < a < 2^k$	$-\frac{R}{2a} + \frac{q}{2} < b' < \frac{R}{2a} - (1 + \frac{1}{a})\frac{q}{2}$

Tabella 5.1: Criteri di selezione dei parametri a e b' a seconda del tipo di ADC e dalla presenza o assenza di arrotondamento. Il passo di quantizzazione q vale $R/2^k$.

risce il valore campionato e codificato dall'ADC al DAC. I due segnali analogici, quello all'ingresso dell'ADC e quello in uscita dal DAC, sono sommati (con segno negativo per il secondo). Nella stessa operazione viene sommata anche la variabile *bias*. Il segnale somma così prodotto viene amplificato della quantità *gain* e riportato in ingresso all'ADC per un mezzo di una linea di ritardo analogica. Quest'ultima è composta da due Sample and Hold (S/H) disposti in serie e comandati dai segnali Φ_1 e Φ_2.

Confrontando l'architettura in Fig. 5.1 con quella riportata nel Cap. 4 si può notare una lieve discrepanza: qui la variabile *bias* (parametro b) viene sommata prima del blocco di guadagno *gain* (parametro a) e non viceversa (come in Fig. 4.7). In tali condizioni si elimina la necessità del blocco di somma del parametro b, aggregando questa operazione a quella già presente a monte del blocco di guadagno. Operando questa trasformazione si ottiene il nuovo parametro

$$b' = \frac{b}{a} \tag{5.1}$$

da cui si ricava

$$b = ab' \tag{5.2}$$

dove b' assume il ruolo della variabile *bias* in Fig. 5.1.

I criteri di selezione dei parametri a e b sono similari a quelli citati precedentemente in Tab. 4.1. Tuttavia, riportiamo nuovamente la tabella (vedi Tab. 5.1), esplicitando il nuovo parametro b', sfruttando la 5.2.

5.2 Hardware

Mostrata l'architettura generale di un TRNG basato su mappa M_{ADC} , abbiamo ora tutti i dati per progettare il dispositivo hardware, focalizzando l'attenzione sugli aspetti fondamentali, quali

- costo ridotto (*low cost*);
- bassa dissipazione di potenza (*low power*);
- throughput ragionevolmente elevato;
- portabilità[9] dell'architettura (*portable*);
- dimensioni e peso ridotto (*low size*).

5.2.1 Selezione dei componenti

Una delle prime fasi nella realizzazione di un dispositivo hardware è quella di *scouting*, nella quale viene stilata una lista di possibili componenti elettronici in grado di rispettare i vincoli di progetto. Tra questi ultimi verrà selezionato quello con il miglior rapporto costo/prestazioni. Di seguito, una selezione dei componenti critici dell'architettura.

5.2.1.1 Microcontrollore e convertitore AD

Il microcontrollore (μC) è l'unità di elaborazione centrale del nostro sistema e deve garantire lo svolgimento delle seguenti funzioni:

- acquisire il segnale analogico relativo alla variabile di stato del sistema;
- pilotare più dispositivi, quali ADC, Programmable Gain Amplifier (PGA) e S/H su bus SPI;
- memorizzare i simboli casuali prodotti;
- effettuare una semplice post-elaborazione a elevato data rate;
- inizializzare e gestire una comunicazione seriale USB.

[9] Intesa come la possibilità di mapparne i blocchi funzionali su μC di differenti costruttori o famiglie con relativo agio.

Codice	Program Memory	RAM	Prezzo
PIC18F14K50	16 KB	768 B	1.53 $
PIC18F24K50	16 KB	2048 B	1.65 $
PIC18F25K50	32 KB	2048 B	1.76 $
PIC18F2550	32 KB	2048 B	3.44 $

Tabella 5.2: Riepilogo dei parametri chiave presenti in Fig. A.1.

In base a quanto appena elencato è evidente che, qualsiasi sia il μC scelto, questo debba possedere i seguenti requisiti minimi:

- un ingresso analogico con ADC, meglio se a risoluzione elevata (8 o più bit);
- un bus dedicato, per comunicare con altri dispositivi (es. I^2C, SPI, ...);
- due uscite digitali, per pilotare i due S/H;
- una sezione di Random Access Memory (RAM) dedicata alla memorizzazione dei simboli prodotti;
- frequenza di lavoro elevata;
- interfaccia USB integrata.

Esperienze pregresse nella programmazione di sistemi a μC e tool di programmazione disponibili al momento della realizzazione di questo scritto, hanno privilegiato la selezione di prodotti della sola famiglia Microchip® PIC®. Tale limitazione non deve essere intesa come vincolo di progetto, in quanto l'architettura del TRNG può essere implementata su qualsiasi piattaforma a μC o μP, che osservi i requisiti minimi esposti sopra.

Per confrontare agevolmente le caratteristiche dei tanti dispositivi che compongono la famiglia PIC®, abbiamo usufruito dello strumento Microchip®Advanced Part Selector (MAPS) fornito dalla stessa Microchip®, vagliando i μC con architettura a 8 bit e prefisso PIC18F, in grado di rispettare i requisiti minimi. Dal risultato della comparazione, visibile in Appendice, Fig. A.1, emergono quattro possibili candidati, come riportanto in Tab. 5.2. Tutti e quattro i μC considerati dispongono dei requisiti minimi esposti sopra e le caratteristiche che li accomunano sono diverse. Le uniche differenze riscontrate, ad esclusione di quella del prezzo,

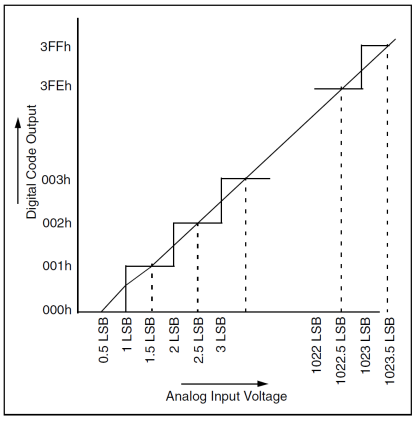


Figura 5.2: Funzione di trasferimento dell'ADC integrato nel PIC18F2550. (fonte [21])

sono quelle relative allo spazio di memorizzazione, per la memoria programma e per la memoria volatile. Nell'ottica dell'obiettivo *low power* tutte e quattro le soluzioni dispongono della tecnologia nanoWatt e nanoWatt XLP™per la riduzione dei consumi energetici.

Anche vagliate tutte queste informazioni, abbiamo optato per il PIC18F2550, poiché comunemente utilizzato nell'ambito della prototipazione elettronica. Questa scelta ha permesso una compilazione rapida del firmware, legata soprattutto alla conoscenza pregressa sulle capacità di questo μC e alla presenza in rete di codice testato (vedi *bootloader* nella sezione 5.3). In ogni caso, i dati raccolti potranno tornare utili in futuro in un possibile contesto di pre-industrializzazione del dispositivo.

L'ADC integrato nel PIC18F2550 ha una risoluzione di 10 bit e uno *swing* compreso nell'intervallo $[V_{SS}, V_{DD}]$. Inoltre, non effettua arrotondamento, come visibile dalla funzione di trasferimento riportata in Fig. 5.2. Le caratteristiche del convertitore e i requisiti di conversione sono consultabili sul datasheet Microchip® [21], tenendo in considerazione anche l'*errata corrige* [22]. In Appendice in Fig. A.2 e in Fig. A.3 sono riportate entrambe per completezza.

5.2.1.2 Convertitore DA

Nella fase di selezione del convertitore DA abbiamo preferito un approccio differente rispetto alla sezione precedente. Sebbene sia stata redatta nuovamente

Codice	Produttore	Risoluzione	Comunicazione	Prezzo
AD5620	Analog®	12 bit	SPI	2.34360 $
DAC101C085	Texas®	10 bit	I^2C	0.73125 $
MAX517	Maxim®	8 bit	I^2C	3.04776 $
MCP4725	Microchip®	12 bit	I^2C	0.73000 $
MCP4911	Microchip®	10 bit	SPI	1.05000 $

Tabella 5.3: Tabella di comparazione delle caratteristiche dei DAC esaminati. (fonte [5])

una lista di possibili candidati DAC, questi sono stati individuati sfruttando un catalogo di componenti elettronici online, in particolare Digikey [5]. I criteri di ricerca applicati sono stati:

- alimentazione singola, siccome il dispositivo deve essere alimentato unicamente per mezzo del bus USB, sul quale è presente la tensione di alimentazione $+5\,\mathrm{V}$;
- un solo convertitore per integrato, con uscita in tensione unipolare;
- risoluzione a 8 o più bit (in quanto l'ADC complementare è a 10 bit);
- comunicazione seriale SPI o I^2C;
- ridotto *settling time*[10], per consentire iterazioni della mappa a elevata frequenza;
- uscita *rail-to-rail*, in modo da sfruttare a pieno la dinamica dell'ADC;
- costo ridotto.

La Tab. 5.3 mostra i cinque possibili candidati emersi dalla ricerca, assieme ai parametri chiave, quali risoluzione, protocollo di comunicazione e prezzo.

Per valutare le prestazioni dei singoli DAC abbiamo costruito il seguente esperimento, ripetuto per ognuno dei cinque candidati:

1. il DAC viene connesso al modulo MSSP del μC (in modalità SPI o I^2C a seconda del protocollo di comunicazione);

[10]Tempo che intercorre tra la conversione DA e il momento in cui il valore è disponibile in uscita, rimanendo confinato entro un intervallo prestabilito.

2. l'uscita del DAC viene portata all'ingresso dell'ADC integrato nel μC;

3. il μC elabora un valore digitale casuale compreso nell'intervallo $[0, 2^k]$, dove k è la risoluzione in bit del DAC sotto esame;

4. il valore casuale generato viene inviato al DAC, il quale lo converte nel valore di tensione corrispondente in uscita;

5. si effettua il campionamento e l'acquisizione dell'ADC;

6. i due valori digitali, quello inviato al DAC e quello letto dall'ADC, vengono trasmessi al computer *host*.

Ripetendo i passi da 3 a 6, si ottiene la costruzione di un grafico di dispersione, nel quale ogni punto ha ascissa pari al valore inviato al DAC e ordinata pari a quello letto dall'ADC. L'obiettivo di tale esperimento era quello di verificare empiricamente quale coppia ADC/DAC risultasse esattamente complementare. A questo scopo, l'andamento del grafico di dispersione deve essere paragonabile a una retta di pendenza unitaria. Nelle Fig. A.4, A.5, A.6, A.7 e A.8 sono mostrati i grafici di dispersione ottenuti per ognuno dei cinque candidati. In particolare, la prima serie (nominata `DAC`) è costituita da punti appartenenti alla retta di pendenza unitaria (ideale). La seconda serie (nominata `ADC`) riporta i punti letti dall'ADC, i quali si discostano dal caso ideale. La terza serie (nominata `DELTA`) mostra graficamente la divergenza tra le prime due. Inoltre, in ogni figura è riportata l'equazione della retta $y = mx + q$ intersecante i punti della serie `ADC`. La bontà della coppia ADC/DAC è determinata principalmente dal valore del coefficiente angolare m, dove $m = 1$ delinea il caso ideale. Quindi, il *gain error* vale $G_{err}\% = 100|1 - m|$.

Confrontando i risultati ottenuti, riportati in Tab. 5.4, possiamo considerare ottimi candidati i DAC Texas® DAC101C085, Maxim® MAX517 e Microchip® MCP4911, in quanto mostrano tutti quanti un gain error molto ridotto, in questa particolare condizione di verifica. Sebbene le prestazioni ottenute siano paragonabili, abbiamo selezionato il Microchip® MCP4911, visto il suo prezzo ridotto (a confronto con il MAX517) e la comunicazione SPI. Quest'ultima è da preferire alla comunicazione I^2C (presente sul DAC101C085) in quanto, sempre in ottica *low power*, consente di ottenere un risparmio energetico.

Codice	Produttore	m	$\|q\|$	G_{err}
AD5620	Analog®	1.0318	6.8905	3.18%
DAC101C085	Texas®	0.9994	1.5074	0.06%
MAX517	Maxim®	1.0004	1.0143	0.04%
MCP4725	Microchip®	1.0074	7.4894	0.74%
MCP4911	Microchip®	1.0002	2.9250	0.02%

Tabella 5.4: Tabella di comparazione del gain error G_{err} relativo ai DAC esaminati. Per completezza, si riportano anche i valori di coefficiente angolare m e offset della retta interpolante i punti dei grafici di dispersione, visibili in Appendice.

Come già anticipato in Tab. 5.3, il DAC MCP4911 ha una risoluzione di 10 bit (sui quali è garantita la monoticità) e un ridotto *settling time*, pari a $4.5\,\mu$s. La tensione di uscita vale allora

$$V_{OUT} = \frac{V_{ref} \times D}{2^{10}} G \tag{5.3}$$

dove D è il valore digitale a 10 bit ricevuto dal μC e G è un guadagno configurabile (di norma settato a 1). Come quasi tutti i DAC considerati necessita di una tensione di riferimento V_{ref}, fornita dall'esterno. Inoltre, la frequenza massima di comunicazione seriale Serial Peripheral Interface (SPI) è di $20\,\mathrm{MHz}$. Per caratteristiche più dettagliate rimandiamo al datasheet del produttore [23].

5.2.1.3 OpAmp per il blocco di somma

Proseguendo nella sintesi dell'architettura mostrata precedentemente in Fig. 5.1, troviamo il blocco di somma. L'operazione deve essere tale da garantire minimo rumore e offset, ottenendo una tensione di uscita pari a

$$V_{sum} = V_{ADC} - V_{DAC} + V_{bias} \tag{5.4}$$

Dal momento che i tre ingressi sono tensioni, risulta conveniente sintetizzare uno stadio sommatore ad amplificatore operazionale (OpAmp), come quello mostrato in Fig. 5.3. Il sommatore così realizzato ha quattro ingressi in tensione, nominati V_1, V_2, V_3 e V_4. Sotto condizioni di idealità dell'OpAmp ($R_{in} \to \infty$ tale che $i^- = 0$,

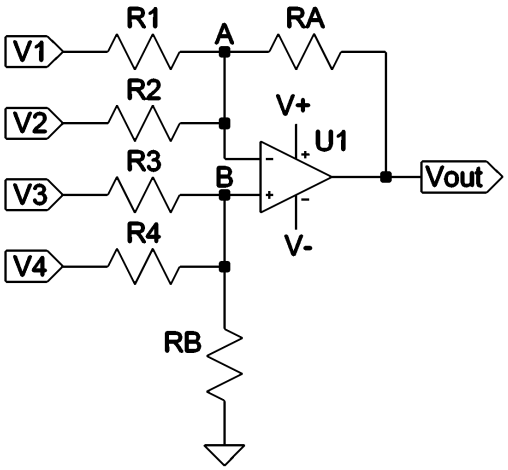


Figura 5.3: Schema elettrico del sommatore ad amplificatore operazionale.

$i^+ = 0$ e $v^- \equiv v^+$) l'equazione di uscita del sommatore vale

$$V_{OUT} = \left(\frac{V_3}{R_3} + \frac{V_4}{R_4}\right) R_A \frac{R_3 \parallel R_4 \parallel R_B}{R_1 \parallel R_2 \parallel R_A} - \left(\frac{V_1}{R_1} + \frac{V_2}{R_2}\right) R_A \qquad (5.5)$$

Imponendo la condizione $R_1 = R_2 = R_3 = R_4 = R_A = R_B$ nella 5.5 otteniamo

$$V_{OUT} = V_3 + V_4 - V_1 - V_2 \qquad (5.6)$$

Confrontando la 5.6 con la 5.4, è evidente che $V_1 \equiv V_{DAC}$, $V_2 \equiv 0$, $V_3 \equiv V_{ADC}$ e $V_4 \equiv V_{bias}$.

In modo analogo a quanto già visto per la selezione del DAC (stesso metodo di ricerca), la valutazione dell'amplificatore operazionale più idoneo deve basarsi sui seguenti requisiti:

- alimentazione singola, siccome il dispositivo deve essere alimentato unicamente per mezzo del bus USB, sul quale è presente la tensione di alimentazione $+5\,\mathrm{V}$;
- ingresso ed uscita *rail-to-rail*, in modo da sfruttare a pieno le dinamiche di ADC e DAC;
- stadio di ingresso di tipo CMOS, in modo da limitare errori *signal dipendent*;
- consumo di corrente ridotto;

- costo ridotto.

Il risultato dello scouting ha evidenziato l'amplificatore Texas® LMV791, il quale rispetta esaurientemente i suddetti requisiti. Riportiamo di seguito alcuni dati caratteristici:

- rumore di tensione riferito all'ingresso: $5.8\,\mathrm{nV}/\sqrt{\mathrm{Hz}}$
- prodotto banda-guadagno: $17\,\mathrm{MHz}$
- corrente assorbita in *Enable Mode*: $1.15\,\mathrm{mA}$
- corrente assorbita in *Shutdown Mode*: $0.02\,\mu\mathrm{A}$

Per informazioni più dettagliate rimandiamo al datasheet del produttore [25].

5.2.1.4 Amplificatore per il blocco di guadagno

Nello stadio di amplificazione il parametro di guadagno a può variare liberamente nell'intervallo $]1, 2^k[$, come già anticipato nel Cap.4. Ancora, è conveniente ricordare che la scelta ottima di a ricade su valori pari alla potenza del due, ovvero $a = 2^i$ con $i = 0, 1, 2, ..., k$. Rapporti di amplificazione di questo tipo ($1:1$, $1:2$, $1:4$, ...) si possono trovare comunemente negli amplificatori destinati alla conversione AD (ne incrementano la risoluzione). In particolare, l'integrato Microchip® MCP6S91 è perfetto al nostro scopo, disponendo di 8 differenti guadagni selezionabili: $+1, +2, +4, +5, +8, +10, +16, +32\,\mathrm{V/V}$. L'MCP6S91 è un Programmable Gain Amplifier (PGA) e dispone di interfaccia di comunicazione SPI, in linea con il protocollo del DAC MCP4911 scelto in precedenza, oltre a ingressi e uscita *rail-to-rail*. Ulteriori dettagli sono presenti nel datasheet del produttore [20].

Il lettore attento potrebbe contraddire la scelta appena operata, in quanto il blocco di guadagno e quello di somma potrebbero essere accorpati in un unico sommatore a OpAmp a guadagno non unitario. Tuttavia, separare la sezione di somma da quella di guadagno comporta due grandi vantaggi:

- possibili errori di guadagno e di offset sui singoli stadi possono essere facilmente corretti poiché disaccoppiati;
- completa flessibilità nella scelta dei parametri a e b.

Quanto appena evidenziato determinerà un processo di prototipazione rapido e lineare nelle sezioni a seguire.

5.2.1.5 OpAmp per il blocco di ritardo

Come già anticipato nella prima sezione di questo capitolo, il blocco di ritardo analogico è realizzato disponendo in cascata due Sample and Hold (S/H), i quali vengono comandati secondo la seguente logica:

1. l'informazione analogica (nel nostro caso una tensione) è presente all'ingresso del primo S/H in modalità *sample*;
2. si commuta la modalità del primo S/H in *hold*, memorizzando l'informazione in ingresso;
3. il secondo S/H, in modalità *sample*, vede l'informazione memorizzata dal primo S/H;
4. commutando anche il secondo S/H in modalità *hold*, si trasferisce l'informazione memorizzata dal primo S/H al secondo S/H.

Ripetendo ciclicamente i quattro passi si ottiene il comportamento cercato, equiparabile a un ritardo analogico di uno *step* temporale.

I S/H possono essere implementati in diversi modi. Consideriamo ora due tipologie di S/H: integrati o a componenti discreti. Il primo tipo viene fornito sotto forma di unico *package*, mentre il secondo è un sottocircuito costituito da due interruttori analogici con abilitazione digitale e un condensatore.

Fanno parte della prima categoria di S/H gli integrati Texas® LF298 e Maxim® DS1843, i quali però non soddisfano i requisiti dell'architettura: l'LF298 necessita di un'alimentazione duale, ottenibile da quella singola solo grazie ad hardware aggiuntivo, mentre il DS1843 ha uno *swing* in ingresso di solo $1\ \mathrm{V}$, riducendo così la capacità *rail-to-rail* ottenuta finora. Sebbene si sia rivelato non idoneo per la nostra particolare applicazione, abbiamo voluto comunque verificare la linearità del DS1843, sfruttando l'esperimento condotto nella sezione 5.2.1.2. In Appendice in Fig. A.10 ne viene mostrato il risultato, ottenuto interponendo il DS1843 tra l'uscita del DAC (l'MCP4911 scelto in precedenza) e l'ingresso dell'ADC integrato nel

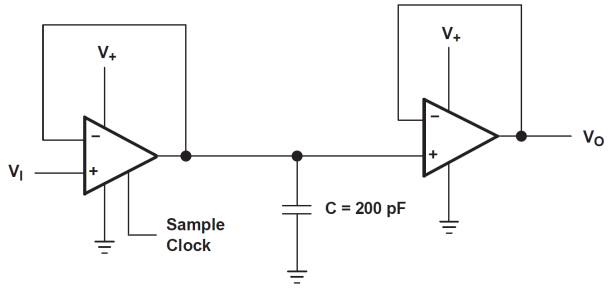


Figura 5.4: Schema elettrico del circuito Sample and Hold. (fonte [26])

μC. Da notare come i valori digitali ottenuti abbiamo un intervallo limitato tra $[0, 205]$ corrispondente all'intervallo di tensioni $[0, 1\,]V$.

Il secondo approccio circuitale è a componenti discreti, come quello riportato in Fig. 5.4. Nell'immagine sono visibili, in cascata, due amplificatori operazionali in configurazione a inseguitore di tensione (buffer) e una capacità intermedia. Mentre l'ultimo buffer (a destra) ha il solo compito di disaccoppiare la carica sulla capacità dai circuiti a valle connessi all'uscita, il primo buffer (a sinistra) ha un ingresso digitale aggiuntivo, spesso denominato *Shutdown* ($\overline{SHDN}$) o *Enable* (EN). Grazie a quest'ultimo è possibile controllare lo stadio di uscita del buffer:

- se $\overline{SHDN} = 1$ il buffer è si trova in modalità attiva e esibisce un comportamento nominale;
- se $\overline{SHDN} = 0$ il buffer è spento e mostra un'impedenza di uscita molto elevata.

Nello stato disabilitato, l'elevata impedenza dello stato di uscita del primo buffer unita all'elevata impedenza d'ingresso del secondo buffer, consentono al condensatore intermedio di non scaricarsi. L'informazione è memorizzata come tensione sul condensatore ed è sempre disponibile in uscita poiché bufferizzata. Il valore della capacità deve essere tale da garantire una costante di tempo di carica $\tau = RC$ ridotta (dove R è la resistenza di uscita del buffer quando attivo), senza compromettere l'integrità dell'informazione memorizzata. Per la nostra applicazione, un ottimo intervallo di valori per C è $100 \div 300\,\mathrm{pF}$.

I criteri per la selezione degli OpAmp sono equiparabili a quelli utilizzati nel caso del blocco di somma, con l'aggiunta di un importante requisito sulla

Codice	Input bias current	GBP	Turn-on time	Prezzo
LMV341	1000 fA	1 MHz	5 μs	0.2900 $
LMV601	20 fA	1 MHz	5 μs	0.2465 $
LMV791	100 fA	17 MHz	140 ns	0.5350 $

Tabella 5.5: Comparazione degli amplificatori operazionali per il blocco di ritardo. (fonte prezzi [5])

corrente di polarizzazione in ingresso, la quale deve essere molto ridotta. Infatti, la progressiva scarica del condensatore è da attribuire in buona parte a quest'ultima: minore è la corrente di polarizzazione in ingresso, minore sarà il degrado dell'informazione memorizzata sulla capacità nel tempo. I dispositivi individuati sul catalogo Digikey [5] sono riportati in Tab. 5.5, nella quale riportiamo i soli codici in quanto tutti prodotti dalla Texas Instruments®. L'LMV791, già considerato per il blocco di somma, è una valido candidato. In particolare dimostra di essere migliore sotto l'aspetto della velocità, con tempi di accensione molto ridotti rispetto ai concorrenti e, quindi, frequenze di campionamento più elevate. Tuttavia, il parametro fondamentale per lo stadio di ritardo rimane la corrente di polarizzazione in ingresso. Quindi, il candidato selezionato è l'LMV601. Per informazioni dettagliate rimandiamo al datasheet del produttore [26].

Per realizzare il blocco di ritardo sono necessari due S/H consecutivi, per un totale di quattro amplificatori operazionali: due provvisti di ingresso di abilitazione e due standard. Per ridurre il numero di componenti e minimizzare lo spazio occupato sulla Printed Circuit Board (PCB), è conveniente sfruttare la controparte con suffisso `XX2` (LMV342, LMV602 e LMV792), nella quale due amplificatori privi di ingresso di abilitazione trovano sede all'interno dello stesso package.

Il comportamento del blocco di ritardo, quando interposto tra DAC e ADC nei test del DAC, è mostrato in Appendice in Fig. A.9. A parte un offset costante, facilmente correggibile via firmware, il blocco di ritardo è sufficientemente lineare nell'intervallo centrale della tensione di alimentazione. Siccome, avvicinandosi al *rail* superiore o inferiore, il comportamento peggiora, sarà conveniente scegliere un IS possibilmente centrato rispetto alla metà della tensione di alimentazione.

5.2.1.6 Tensione di riferimento e bias

Come dichiarato nella sezione 5.2.1.2, il DAC MCP4911 non ha a disposizione una proprio riferimento di tensione, il quale deve essere generato da un componente esterno. A questo scopo, è necessario individuare un riferimento di tensione a basso costo in grado di erogare una tensione stabile e nota. Il componente individuato appartiene alla famiglia Texas® MAX600X, la quale comprende i valori $1.25\,\mathrm{V}$, $2.5\,\mathrm{V}$, $3\,\mathrm{V}$, $4.096\,\mathrm{V}$ e $5\,\mathrm{V}$. Il valore selezionato per la nostra applicazione è $4.096\,\mathrm{V}$, contraddistinto dal codice MAX6004. Tale valore garantisce al regolatore un funzionamento nominale (tenendo conto di una ridotta caduta di tensione) anche nel caso in cui la tensione di alimentazione V_{BUS} sia inferiore allo standard $5\,\mathrm{V}$. Ulteriori informazioni sono riportate sul datasheet del produttore [27].

Ulteriore impiego del riferimento di tensione è la generazione di V_{bias}, individuata nella sezione 5.2.1.3. Quest'ultima corrisponde al parametro b' dell'architettura mostrata in Fig. 5.1. Per ottenere b', che è solo una frazione del riferimento, si sfrutta un classico partitore resistivo, disaccoppiato dalla circuiteria a valle mediante un buffer analogico, realizzato con l'integrato Texas® LMV341 [24]. Inoltre, per uniformità, lo stesso riferimento è utilizzato dall'ADC interno al μC.

5.2.2 Prototipo di TRNG basato su dinamica caotica

Nella Fig. 5.1 mostrata all'inizio del capitolo è visibile l'architettura generale del TRNG oggetto della proposta di tesi. Tuttavia questa rappresentazione non fornisce il livello di dettaglio necessario in questa fase. A questo scopo, è conveniente fare riferimento alla Fig. 5.5, nella quale sono esplicitati i blocchi logici trattati precedentemente e le interconnessioni analogiche e digitali tra di essi. In particolare, tra i segnali digitali possiamo distinguere il bus USB tra μC e computer *host*, il bus SPI per la comunicazione con ADC e PGA e i segnali di controllo dei S/H, `SHEN1` e `SHEN2`.

Ora, nell'architettura evidenziata in Fig. 5.5 possono trovare collocazione tutti i componenti identificati della sezione precedente:

- μC: Microchip® PIC18F2550
- DAC: Microchip® MCP4911

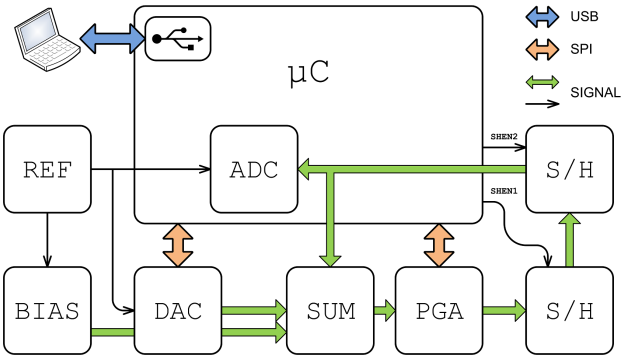


Figura 5.5: Schema a blocchi del generatore TRNG basato su dinamica caotica.

- SUM: Texas® LMV791
- PGA: Microchip® MCP6S91
- S/H: Texas® LMV601 ($2\times$) e LMV602
- REF: Maxim® MAX6004
- BIAS: Texas® LMV341

Sfruttando la rappresentazione di Fig. 5.5 e la selezione dei componenti, siamo ora in grado di progettare lo schema elettrico del prototipo di TRNG basato su dinamica caotica, mostrato in Appendice in Fig. A.11.

Al fine di ottenere una piattaforma performante e flessibile, sono stati adottati i seguenti accorgimenti:

- condensatori di *bypass*: piazzati in prossimità dei piedini di alimentazione dei singoli integrati, consentono una notevole riduzione del rumore dovuto a disturbi e interferenze;
- oscillatore al quarzo: unito al Phase-Locked Loop (PLL) interno al μC garantisce una velocità di eleborazione elevata, pari a $12\,\mathrm{Mips}$;
- riduzione dei consumi: in ottica *low power*, segnali di controllo digitali permettono al μC di poter spegnere i singoli integrati, minimizzando la corrente assorbita dal TRNG (lo stesso μC può essere configurato in modalità *Deep Sleep*, limitando notevolmente i propri consumi);

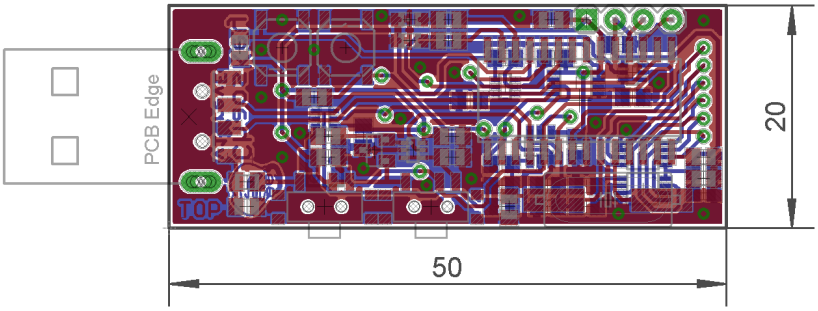


Figura 5.6: Realizzazione CAD del prototipo TRNG basato su dinamica caotica.

- filtraggio dell'alimentazione: come raccomandato dalle specifiche sullo standard USB, condensatori di disaccoppiamento e induttanze (ferrite) sono stati aggiunti al percorso di alimentazione V_{BUS};
- selettore del riferimento: un interruttore consente di selezionare quale tensione di riferimento impiegare tra le due disponibili, l'alimentazione V_{BUS} a $5\,\mathrm{V}$ o il riferimento a $4.096\,\mathrm{V}$ generato dal MAX6004;
- pulsante di RESET: per re-inizializzare il sistema allo stato nominale (in caso di malfunzionamenti);
- pulsante di BOOT: per accedere alla modalità *bootloader* (vedi sezione 5.3);
- luci di segnalazione: per fornire all'utente un riscontro visivo immediato sullo stato del sistema.

Ottenuto lo schema elettrico in Fig. A.11, siamo ora in grado di progettare, sfruttando un Computer Aided Desing (CAD) elettronico, un prototipo tangibile di TRNG basato du dinamica caotica. In Fig. 5.6 ne è mostrato un esempio.

Le dimensioni minime del prototipo ($50 \times 20\,\mathrm{mm}$, escluso il terminale USB) sono paragonabili alle dimensioni standard per *dongle* USB commerciali. La riduzione del formato fisico del dispositivo ne aumenta la praticità, oltre a ridurre i costi di produzione della PCB. Inoltre, Solamente la selezione di componenti in tecnologia Surface Mount Technology (SMT) ha consentito di ottenere tale risultato. In Fig. 5.7 è visibile una realizzazione tangibile funzionante del prototipo di TRNG basato su dinamica caotica. Grafici e risultati presenti nel prossimo capitolo sono stati valutati sul suddetto hardware.

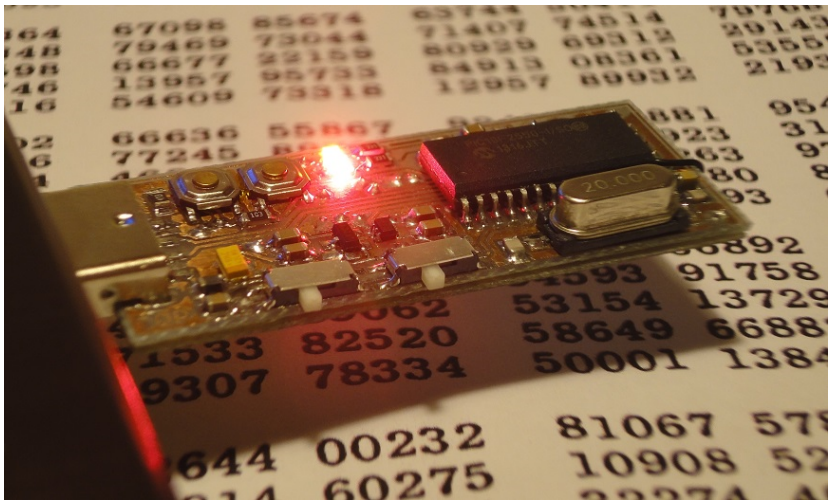


Figura 5.7: Prototipo tangibile funzionante di TRNG basato su dinamica caotica.

5.2.2.1 Prospetto economico

Un aspetto molto importante da tenere in considerazione nella progettazione di un prodotto elettronico è quello economico. Sul mercato sono già presenti TRNG commerciali, sebbene nessuno di questi sfrutti le dinamiche caotiche come sorgente di rumore. Quindi, è risultato conveniente, già a partire da questa prima sperimentazione, rendere il generatore oggetto di questo elaborato competitivo, sia a livello economico che prestazionale. A tale scopo, in Tab. 5.6, è riportato un prospetto di prezzi e quantità dei singoli componenti, comunemente denominato Bill Of Material (BOM), in modo tale da avere almeno una panoramica generale sul costo finale della piattaforma proposta.

5.2.2.2 Future ottimizzazioni

All'inizio del capitolo è stata fatta una precisazione sull'architettura del sistema relativa alla posizione del blocco di somma del parametro b, il quale è stato portato a monte del blocco di guadagno a. Come sopracitato si identifica il nuovo parametro b' (vedi 5.1). Solo una volta realizzato il prototipo hardware e verificato sperimentalmente il suo comportamento è stato possibile concludere che tale modifica ha portato più svantaggi che benefici. La scelta operata all'inizio del capitolo è stata compiuta unicamente per cercare di ridurre il numero di componenti presenti

Qty	Value	Device	Parts	Unit Price [$]	Extended Price [$]
6	100K	Resistor	R6, R7, R8, R9, R11, R12	0.00231	0.01386
12	100nF	Ceramic capacitor	C1, C6, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19	0.0721	0.8652
1	10K	Trimmer resistor	R1	0.67488	0.67488
2	10K	Resistor	R2, R3	0.00231	0.00462
1	10nF	Ceramic capacitor	C9	0.0721	0.0721
1	1nF	Ceramic capacitor	C5	0.0721	0.0721
2	200pF	Ceramic capacitor	C7, C8	0.01336	0.02672
1	20MHz	Crystal	Q1	0.176	0.176
2	22pF	Ceramic capacitor	C2, C3	0.01336	0.02672
2	330	Resistor	R4, R5	0.00231	0.00462
1	470nF	Ceramic capacitor	C4	0.04286	0.04286
2	ACT, BOOT	Led	LED1, LED2	0.05589	0.11178
1	BLM21	Ferrite bead	FER1	0.0168	0.0168
2	LMV341	OpAmp	IC1, IC3	0.29	0.58
2	LMV601	OpAmp w/Shutdown	IC4, IC6	0.2465	0.493
1	LMV602MM	OpAmp	IC5	0.3103	0.3103
1	LMV791	OpAmp w/Shutdown	IC9	0.535	0.535
1	MAX6004	Voltage reference	IC2	0.62424	0.62424
1	MCP4911MS	DAC 10-bit with SPI	IC7	1.05	1.05
1	MCP6S91MS	PGA with SPI	IC8	0.65	0.65
1	PIC18F2550	PIC microcontroller	U1	3.97999	3.97999
2	POWER, REF	Switch SPDT	SL1, SL2	0.2944	0.5888
2	RESET, BOOT	Switch momentary	S1, S2	0.0841	0.1682
1	USB A	USB-A male connector	JP2	1.67	1.67
				Total Price [$]	12.75779

Tabella 5.6: BOM relativa allo schema elettrico in Fig. A.11. (fonte prezzi [5])

nella BOM. Così facendo si risparmia l'ulteriore amplificatore operazionale e i componenti passivi annessi necessari per il blocco di somma del parametro b, aggregando questa operazione a quella già presente per la differenza. Sebbene da un lato abbiamo una riduzione nel numero dei componenti elettronici necessari, dall'altro abbiamo dato origine a due possibili problemi:

- come ogni altra grandezza elettrica del circuito, anche b è soggetta a rumore. Di conseguenza, secondo la 5.1, anche il parametro b' sarà soggetto a rumore, amplificato però di a volte.
- ad essere fissato da progetto, in questa configurazione, è il parametro b'. Dalla 5.2, si ottiene una riduzione del valore di b di a volte.

Soprattutto nel caso in cui a è elevato, risulta complessa e rumorosa l'implementazione hardware del riferimento b'. Quindi, è doveroso tenere conto di queste considerazioni in future ottimizzazioni.

Inoltre, lo schema circuitale in Fig. A.11, poiché non ottimizzato per la preindustrializzazione, consente la regolazione dei parametri liberi dell'architettura. In particolare, è possibile modificare via software il parametro a, avendo preferito un PGA a un operazionale a guadagno fisso, o variare il parametro b' regolando la resistenza variabile R1. In tal modo, è garantita un'elevata flessibilità in fase di sperimentazione, grazie alla quale è possibile verificare le prestazioni di differenti configurazioni circuitali. Tuttavia, tali regolazioni richiedono l'intervento dell'operatore e devono pertanto essere evitate in una possibile fase di produzione industriale. Quindi, una volta identificati sperimentalmente i valori ottimi per questa particolare implementazione hardware, è indispensabile vincolare da progetto tali parametri liberi. A questo scopo, è possibile, ad esempio, sostituire la resistenza variabile con due resistori fissi di rapporto uguale e rimpiazzare il PGA con l'equivalente realizzazione ad amplificatore operazionale.

5.3 Firmware

Ponendosi come strato intermedio tra *hardware* e *software*, il *firmware* deve interagire con i componenti elettronici, fornendo allo stesso tempo un livello di

astrazione superiore lato operatore. A questo scopo, il codice in esecuzione sul μC deve garantire svariate funzioni, unificabili in cinque fasi distinte:

- inizializzazione: viene stabilita la comunicazione USB e si provvede alla configurazione delle periferiche interne (ADC) ed esterne su bus SPI (DAC e PGA);
- controllo: vengono scambiati i parametri di controllo con l'applicazione lato utente;
- generazione: il sistema caotico viene portato in auto-oscillazione, durante la quale i simboli casuali prodotti vengono memorizzati localmente;
- post-processing: opzionalmente, i simboli casuali vengono elaborati in modo tale da produrre sequenze di bit IID, a discapito del data rate;
- trasmissione: i bit *random* vengono aggregati in pacchetti e inviati all'applicazione lato utente.

Tra quelle appena citate, le fasi rilevanti per il livello di approfondimento di questa trattazione sono quella di generazione e post-processing. Il meta-codice relativo alla fase di generazione è riportato di seguito e commentato blocco per blocco.

```
loop()
{
    SHEN2 = 1;                    // track state
    Delay10TCYx( 6 );             // turn-on delay from shutdown (5us)

    SHEN2 = 0;                    // hold state
    Delay10TCY();                 // little delay to ensure S/H is in hold state (0.83us)
```

Si esegue il campionamento sul secondo S/H, seguito da memorizzazione. L'informazione viene trasferita tra il primo e il secondo S/H, realizzando così un ritardo temporale pari a un tempo di ciclo.

```
    ConvertADC();                 // start ADC acquisition
    while( BusyADC() );           // wait for ADC conversion
    adc_read = ReadADC();         // read ADC value
```

Una volta avviata e conclusa l'acquisizione dell'ADC interno al μC, il valore campionato viene memorizzato nella variabile *adc_read*. Il campione è la rappresentazione quantizzata (a 10 bit) della variabile di stato del sistema.

```
write_MCP4911( adc_read & bitmask );    // write most significant bit to DAC
```

Il valore letto in precedenza viene inviato a mezzo SPI al DAC, il quale lo converte nella rispettiva tensione alla sua uscita. In realtà solo alcuni bit più significativi (tra i 10 disponibili) vengono utilizzati. Quanti e quali siano i bit da scartare è stato configurato nella precedente fase di controllo. Consideriamo k la risoluzione nominale del DAC. Se il valore ricevuto ha l bit meno significativi vincolati a zero, la risoluzione reale viene ridotta a $k - l$ bit. Questo comporta una riduzione nel numero di rampe che compongono il segnale errore $E(x)$ (vedi Sez.4.2). In particolare, se nominalmente avremmo avuto 2^k rampe, ora $E(x)$ è formata da sole 2^{k-l} rampe. Quindi, a parità di a, il numero di rami che costituiscono la mappa M_{ADC} sono stati ridotti di un fattore 2^l. Tale effetto si unisce ai parametri liberi a e b', divenendo un'ulteriore variabile di controllo sul sistema caotico. Idealmente, a e l sono interscambiabili: raddoppiare a (che ricordiamo essere una potenza di 2) è equivalente, ai fini della struttura della mappa, a ridurre l di una unità.

```
SHEN1 = 1;                  // track state
Delay10TCYx( 6 );           // turn-on delay from shutdown (5us)

SHEN1 = 0;                  // hold state
Delay10TCY();               // little delay to ensure S/H is in hold state (0.83us)
```

Ovviamente, la sezione analogica del circuito, interposta tra DAC e S/H, è tempo continua. Perciò, all'ingresso del primo S/H è già presente la differenza tra la variabile di stato e la stessa quantizzata, opportunamente influenzata dai parametri a e b'. Il parametro a è quello selezionato nella fase di controllo, configurato su PGA per mezzo del protocollo SPI. Quindi, in modo simile a quanto visto sopra, si esegue il campionamento sul primo S/H, seguito da memorizzazione. Nel prossimo ciclo, l'informazione memorizzata diventerà la nuova variabile di stato del sistema.

```
switch( adc_read >> 7 )
{
```

```
        case 2:
            temp = 0b00000001;
            break;
        case 3:
            temp = 0b00000010;
            break;
        case 4:
            temp = 0b00000011;
            break;
        default:
            temp = 0b00000000;
    }
```

Come già citato, la variabile *adc_read* è la rappresentazione quantizzata (a 10 bit) della variabile di stato del sistema. A seconda della configurazione ricevuta nella fase di controllo, si procede alla memorizzazione dei soli bit contenenti i codici binari che identificano i differenti stati di Markov del sistema caotico. Nell'esempio riportato, gli stati della catena di Markov incorporata nel sistema caotico sono quattro, identificati dai codici binari 00, 01, 10 e 11. Questi ultimi possono essere ottenuti sfruttando i soli tre bit più significativi della variabile *adc_read*, esaminando la struttura dell'istogramma in Fig. 5.8. Quindi, lo schema di codifica sarà 00-01-10-11-00.

```
    if( word_index == 0 )        // first word (2 bits)
    {
        USB_Out_Buffer[ buffer_index ] = temp;
    }
    else                        // second, third and fourth words (shifted)
    {
        USB_Out_Buffer[ buffer_index ] |= temp << ( 2 * word_index );
    }
```

Seguendo ancora l'esempio, i due bit ottenuti ad ogni ciclo sono raggruppati a gruppi di quattro a formare parole da $1\,\mathrm{byte}$, memorizzate localmente in un buffer lungo $64\,\mathrm{byte}$, il quale, una volta riempito, viene inviato all'applicazione lato utente nella fase di trasmissione.

La fase di post-processing è opzionale poiché i bit casuali generati dal sistema caotico possiedono già buone caratteristiche statistiche. Tuttavia, tale funzione è in grado di garantire la rimozione di possibili correlazioni residue e *bias*[11]. A

[11] Sbilanciamento statistico nella distribuzione di *0* e *1*.

questo scopo, l'algoritmo impiegato per il post-processing dei simboli prodotti dal generatore è lo stesso già evidenziato nella Sez. 3.2.1 in Tab. 3.1. I risultati dell'applicazione del post-processing saranno riportati successivamente nella Sez. 5.4.1 e ancora nel Cap. 6.

Il compilatore adottato per la stesura del codice firmware è quello proprietario Microchip®: MPLAB®C18. Inoltre, lo stesso produttore fornisce il codice per l'emulazione seriale USB CDC e il codice relativo al *bootloader*. Quest'ultimo in particolare consente di caricare un nuovo codice compilato sul μC per mezzo della stessa connessione USB impiegata per la trasmissione dei dati, rendendo la piattaforma indipendente da un programmatore esterno (ad esclusione del primo caricamento dello stesso bootloader).

5.4 Software

Il processo di redazione del software lato operatore ha avuto un percorso parallelo a quello della compilazione del firmware. Come già anticipato in precedenza, la modalità scelta per il trasferimento dei dati dal TRNG al computer è l'emulazione seriale USB CDC. Grazie a quest'ultima la complessità del codice lato operatore è ridotta, paragonabile alla lettura/scrittura seriale su file. Inoltre, la classe USB CDC è pienamente compatibile con la maggior parte dei Sistema Operativo (SO), rendendo il codice lato macchina sufficientemente portabile.

5.4.1 Elaborazione dati su Matlab®

In ambiente Matlab® è possibile richiamare la lettura/scrittura seriale mediante le funzioni `fopen`, `fread`, `fwrite` e `fclose`. Grazie a queste ultime si inizializza, gestisce e conclude la comunicazione con il TRNG. Al fine di ottenere una visualizzazione grafica della mappa M_{ADC} realmente implementata, sulla quale si basa la dinamica caotica del TRNG, è necessario avere a disposizione i valori letti dall'ADC in formato esteso a $10\,\mathrm{bit}$. Quindi, la costruzione della mappa avverrà per punti, dove il singolo punto ha ordinata pari al valore attuale e ascissa pari al valore precedente. Come evidenziato a sinistra in Fig. 5.8, la morfologia della mappa è equivalente a quella teorizzata nel Cap. 4.

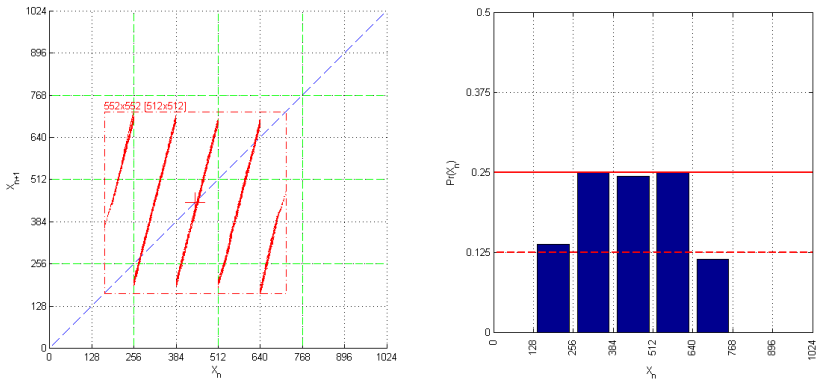


Figura 5.8: Interfaccia di ricezione dei dati in Matlab®. A sinistra, la visualizzazione della mappa M_{ADC} a 4 rami ottenuta utilizzando i parametri $k = 10\,\text{bit}$, $k - l = 10 - 7 = 3\,\text{bit}$, $a = 4$ e $b = \frac{384}{1024}V_{ref}$ (ovvero $b' = \frac{b}{a} = \frac{96}{1024}V_{ref}$). A destra, l'istogramma relativo alla probabilità di trovarsi su ciascuno dei rami della mappa ($\Pr\{X_n\} = \frac{1}{4}$ per i tre rami centrali e $\Pr\{X_n\} = \frac{1}{8}$ per i due rami agli estremi).

Per ottenere la suddetta mappa M_{ADC} a 4 rami sono stati utilizzati i parametri:

- risoluzione nominale del DAC, $k = 10\,\text{bit}$;
- risoluzione reale del DAC, $k - l = 3\,\text{bit}$, dove $l = 7\,\text{bit}$ scartati;
- guadagno dell'amplificatore (PGA), $a = 4$;
- offset dopo l'amplificatore, $b = \frac{384}{1024}V_{ref}$ ($b' = \frac{b}{a} = \frac{96}{1024}V_{ref}$).

Grazie allo script Matlab® è possibile valutare visivamente la qualità generale dell'architettura. In particolare, possiamo notare la limitata dispersione dei punti che compongono i rami della mappa, indice di un basso livello di rumore, oltre alla lunghezza di tali rami, i quali rimangono confinati all'interno dell'Invariant Set. Nella stessa Fig. 5.8 è riportato l'istogramma relativo agli intervalli di partizione $\{X_n\}$, i quali rappresentano la probabilità di trovarsi su uno dei rami della mappa. Ovviamente, considerati i parametri utilizzati e la topologia della mappa M_{ADC} , la probabilità di trovarsi su uno dei rami centrali è $\Pr\{X_n\} = \frac{1}{4}$, pari al 25%, mentre è $\Pr\{X_n\} = \frac{1}{8}$ sui due mezzi rami estremi, pari al 12.5%. Se lo schema di codifica utilizzato è quello adottato in teoria, gli *slot* saranno contraddistinti dai codici 00-01-10-11-00. Considerate le probabilità appena citate, è evidente l'equiprobabilità dei quattro codici, da cui $\Pr\{\texttt{00}\} \equiv \Pr\{\texttt{01}\} \equiv \Pr\{\texttt{10}\} \equiv \Pr\{\texttt{11}\} = \frac{1}{4}$.

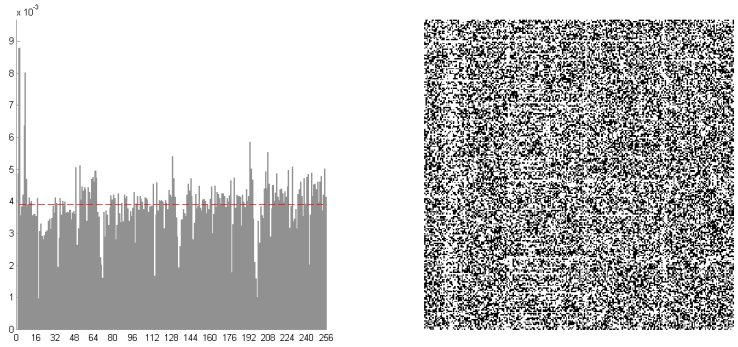


Figura 5.9: Statistiche sulla sequenza binaria generata dal TRNG in assenza di post-processing. A sinistra, la distribuzione dei valori relativi alla segmentazione della sequenza in blocchi di $1\,\mathrm{Byte}$ risulta non uniforme. A destra, l'immagine di distribuzione spaziale, costruita effettuando un'operazione di XOR tra byte successivi, presenta dei *pattern*, imputabili a correlazioni residue.

Allo scopo di rendere visivamente fruibile la qualità delle sequenze prodotte dal TRNG, è utile costruire un'immagine di distribuzione spaziale, generata da un secondo applicativo Matlab®. Mostrata sulla destra nell'esempio di Fig. 5.9, questa rappresentazione permette di individuare istantaneamente la presenza di *pattern* (l'occhio umano è particolarmente efficiente nel rilevare strutture complesse), i quali sono imputabili a correlazioni nella sequenza prodotta. La costruzione dell'immagine, di dimensioni $256 \times 256\,\mathrm{pixel}$, avviene per pixel binari (B/N), adottando come coordinate

- `pixelX`: valori nell'intervallo $[0, 255]$ ottenuti dividendo la sequenza binaria prodotta dal TRNG in blocchi da $8\,\mathrm{bit} \equiv 1\,\mathrm{Byte}$;
- `pixelY`: valore assunto al passo precedente da `pixelX`.

Sotto queste condizioni, l'immagine creata fornisce un riscontro visivo sulla correlazione temporale tra gruppi di bit e la loro distribuzione spaziale. Scorrendo l'intera sequenza (bit per bit) e effettuando un'operazione di XOR tra il valore del pixel calcolato e quello precedente, quella che si ottiene è una mappa di pixel distribuiti nel piano. Idealmente, una distribuzione spaziale uniforme è attribuibile ad un processo di tipo Gaussiano (rumore bianco). Nel caso reale, osservando l'immagine di distribuzione spaziale in Fig. 5.9, relativa alla sequenza prodotta

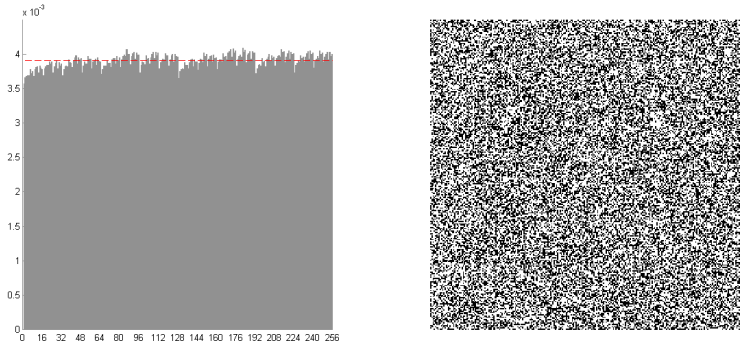


Figura 5.10: Statistiche sulla sequenza prodotta dal TRNG in assenza di post-processing, adottando la codifica alternativa 01-00-10-11-01. A sinistra, la distribuzione dei valori relativi alla segmentazione della sequenza in blocchi di $1\,\mathrm{Byte}$ è quasi uniforme. A destra, l'immagine di distribuzione spaziale, costruita effettuando un'operazione di XOR tra byte successivi, non mostra *pattern* evidenti.

dal TRNG in assenza di post-processing, possiamo notare la presenza di *pattern*, ovvero di zone in cui la distribuzione spaziale dei punti non è uniforme. Ciò è imputabile a correlazioni residue o *bias* nella sequenza binaria generata.

Allo scopo di ridurre tali contributi è stato ideato lo schema di codifica alternativo 01-00-10-11-01, grazie al quale si riduce la sensibilità del generatore a sbilanciamenti nella distribuzione dei codici. Infatti, osservando lo schema di codifica classico 00-01-10-11-00, è possibile notare una differenza nella distribuzione di `0` e `1` (i primi in numero maggiore nella zona centrale). Differentemente, lo schema di codifica alternativo 01-00-10-11-01, in media, è più bilanciato, mostrando una distribuzione di `0` e `1` più uniforme. Ciò permette di ottenere prestazioni migliori, come mostrato in Fig. 5.10, la quale non mostra *pattern* evidenti.

Tuttavia, la distribuzione statistica (a sinistra) in Fig. 5.10 non è ancora perfettamente uniforme, sebbene migliorata rispetto al caso precedente in Fig. 5.9. Per ovviare anche a questo problema può essere sfruttato un algoritmo di post-processing, come quello di Von Neumann, mostrato in precedenza in Tab. 3.1, grazie al quale otteniamo un'immagine di distribuzione spaziale uniformemente distribuita, visibile in Fig. 5.11. La stessa rappresentazione può essere paragonata a rumore bianco. A conferma di quest'ultima osservazione, si noti la perfetta uniformità dell'istogramma della distribuzione statistica.

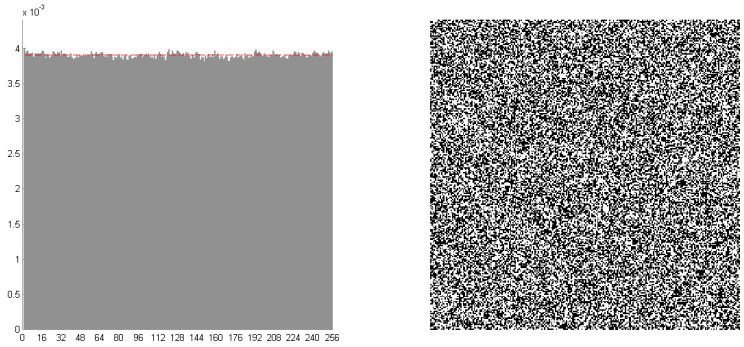


Figura 5.11: Statistiche sulla sequenza prodotta dal TRNG, adottando la codifica alternativa 01-00-10-11-01 e processando i simboli mediante algoritmo di Von Neumann (vedi 3.1). A sinistra, la distribuzione dei valori relativi alla segmentazione della sequenza in blocchi di $1\,\mathrm{Byte}$ è praticamente uniforme. A destra, l'immagine di distribuzione spaziale, costruita effettuando un'operazione di XOR tra byte successivi, mostra uniformità spaziale (rumore bianco).

Ulteriori ed approfonditi test statistici sulla qualità della sequenza binaria generata dal TRNG sono riportati in Appendice e commentati nel Cap. 6.

5.5 Fruizione sul sistema operativo Linux™

Linux™, nelle sue diverse distribuzioni, è attualmente il Sistema Operativo (SO) più utilizzato in applicazioni embedded riconducibili al *mondo web* e *Internet of Things* [29]. Alcune possibili piattaforme basate su SO Linux™ sono

- sistemi a μP ARM® (su distribuzione Debian);
- dispositivi mobile (su distribuzione Android™);
- access point (su distribuzione OpenWRT);
- home media player;
- ...

È evidente quindi che la fruizione in ambiente Linux™ è particolarmente importante in termini applicativi. Perciò, non deve sorprendere la scelta del sistema

operativo Linux™ come piattaforma su cui sperimentare le prestazioni del TRNG oggetto di questa tesi.

5.5.1 Linux™RNG

La generazione di numeri casuali sui sistemi operativi Linux™ è suddivisibile in tre fasi distinte:

- collezionamento: il SO colleziona entropia proveniente da vari eventi del kernel;
- miscelamento: l'entropia raccolta è aggiunta all'entropy pool primario, miscelandola a quella già presente tramite una *mixing function* di tipo Twisted General Feedback Shift Register (TGSFR);
- estrazione: in seguito alla richiesta di bit casuali, si estrae entropia dall'entropy pool primario, generando numeri casuali negli entropy pool secondari.

Onde evitare di lasciare trapelare all'esterno informazioni sullo stato interno dell'entropy pool primario, in uscita dal Linux Random Number Generator (LRNG) viene impiegata la funzione di hash SHA-1 [16], la quale, come ogni altro algoritmo di hash, produce un messaggio *digest* di lunghezza fissa partendo da un messaggio di lunghezza variabile. La sicurezza dello stato interno dell'entropy pool è garantita dall'irreversibilità della funzione di hash, in quanto è impossibile risalire al messaggio originale conoscendo il solo digest prodotto. In Fig. 5.12 è rappresentata la schematizzazione di un LRNG, nel quale sono evidenziate, da sinistra a destra, le sorgenti di entropia del kernel, l'entropy pool primario e i due entropy pool secondari. Gli eventi casuali sfruttati all'interno del kernel sono generalmente quattro:

- tempi di accesso a dispositivi bloccanti (come l'HDD);
- occorrenze di interrupt di sistema;
- digitazione sulla tastiera;
- movimenti e pressioni dei pulsanti del mouse.

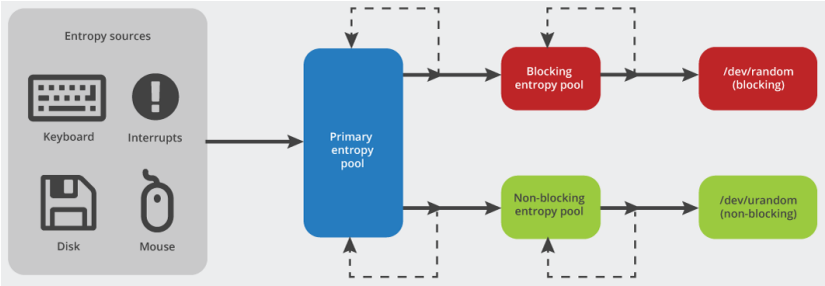


Figura 5.12: Schematizzazione dell'entropy pool di Linux™. (fonte [54])

Mentre le prime due sorgenti sono imputabili a processi deterministici, vulnerabili quindi a possibili attacchi, le ultime due sorgenti sono basate su interazioni con operatori umani, le quali vengono generalmente considerate casuali (sebbene queste non siano intrinsecamente dovute a processi non deterministici) [3].

Le applicazioni utente che necessitano di numeri casuali possono accedere a una delle due interfacce `dev/random` o `dev/urandom`, le quali attingono direttamente dagli entropy pool secondari. La prima interfaccia è di tipo bloccante, a differenza della seconda. Se l'entropia contenuta nell'entropy pool primario è inferiore a quella richiesta, `dev/random` si arresta, in attesa di nuova entropia dalle sorgenti. Diversamente, `dev/urandom` fornisce numeri casuali indipendentemente dal livello di entropia dell'entropy pool primario, impiegando ricorsivamente la funzione di hash SHA-1. Inoltre, tre contatori mantengono aggiornata una stima del livello di entropia contenuta nei diversi entropy pool [42].

Sono evidenti allora le peculiarità delle due differenti interfacce, le quali devono essere impiegate nel giusto ambito: si deve prediligere `dev/random` in applicazioni in cui è richiesta un'elevata qualità (es. crittografia a chiave pubblica), mentre si può sfruttare `dev/urandom` quando è necessario un maggiore data rate, seppure a minore qualità.

Sfortunatamente, su sistemi embedded o server, non si verificano attività umane. Inoltre, quasi tutti i sistemi embedded sono sprovvisti di dispositivi di memorizzazione a testina magnetica (es. HDD), in quanto la memoria fisica del sistema è spesso affidata a logiche allo stato solido (flash memory). Queste ultime hanno tempi di accesso molto più predicibili e costanti, non adoperabili come sorgenti di entropia. In tali condizioni, soprattutto su sistemi embedded, il LRNG è privato di

quasi tutte le sue sorgenti e l'entropy pool raramente raggiunge livelli accettabili di entropia. Su questi sistemi la generazione di numeri casuali è affidata in gran parte alle sole funzioni di mixing e di hash, risolvendo il problema della sicurezza mediante la complessità computazionale.

Per rendere intrinsecamente sicuri i sistemi descritti sopra, può essere conveniente adottare sorgenti di entropia esterne al kernel, come HRNG integrati nei μP o TRNG aggiuntivi (vedi esempi riportati nel Cap.3).

5.5.2 Implementazione del demone su Linux™

Per sfruttare l'entropia prodotta dal TRNG oggetto della proposta di tesi, è necessario progettare un applicativo *ad hoc*, il cui unico scopo è mantenere alto il livello di entropia nell'entropy pool primario, trasferendovi i valori casuali ricevuti sull'interfaccia seriale `dev/ttyACM0`. Consultando le pagine del manuale di Linux™ (in particolare la sezione `random(4)`), si evidenzia la possibilità di dialogare con il LRNG attraverso l'interfaccia `ioctl(2)`. Quest'ultima prevede le operazioni

- `RNDGETENTCNT`, restituisce il contatore dell'entropia presente nel pool primario (lo stesso valore può essere ottenuto mediante la lettura di `/proc-/sys/kernel/random/entropy_avail`);
- `RNDADDTOENTCNT`, incrementa o decrementa il contatore dell'entropia presente nel pool primario;
- `RNDADDENTROPY`, aggiunge entropia al pool primario, incrementandone il relativo contatore;
- `RNDZAPENTCNT` o `RNDCLEARPOOL`, azzera l'entropia di tutti gli entropy pool.

L'operazione `RNDADDENTROPY`, sulla quale è incentrato l'applicativo, utilizza la struttura

```
struct rand_pool_info{
    int        entropy_count;
    int        buf_size;
```

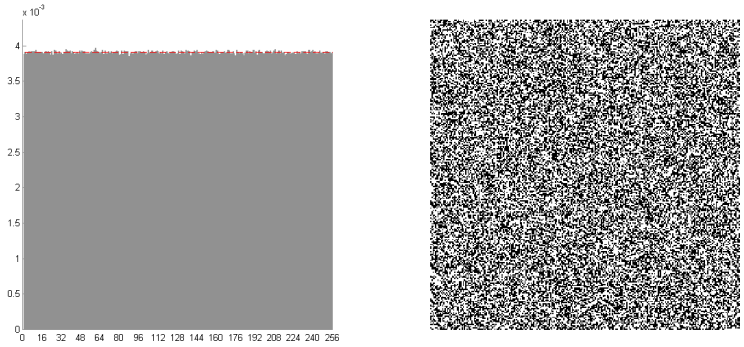


Figura 5.13: Statistiche sulla sequenza prelevata in uscita dal distiller di Linux™, quando il TRNG viene impiegato come sorgente di rumore. A sinistra, la distribuzione dei valori relativi alla segmentazione della sequenza in blocchi di $1\,\mathrm{Byte}$ è uniforme. A destra, l'immagine di distribuzione spaziale, costruita effettuando un'operazione di XOR tra byte successivi, mostra la tipica uniformità spaziale Gaussiana (comparabile a rumore bianco).

```
        int32       buf[0];
    };
```

dove `entropy_count` è il valore aggiunto o sottratto al contatore del pool primario (espresso in bit di entropia) e `buf` è un array di valori a $32\,\mathrm{bit}$ di lunghezza `buf_size` (espresso in byte). Inoltre, è importante ricordare che per effettuare tali operazioni, ad esclusione di `RNDGETENTCNT`, sono necessari privilegi di `root`.

All'ultima versione di Linux™, la dimensione dell'entropy pool primario, riportata anche nel file `/proc/sys/kernel/random/poolsize`, è pari a $512\,\mathrm{Byte}$, ovvero $4096\,\mathrm{bit}$. Quindi, considerato che il TRNG invia pacchetti di $64\,\mathrm{Byte}$, sono sufficienti solamente 8 trasmissioni consecutive per riempire completamente l'entropy pool primario. Quanto appena detto vale solo nel caso ideale, ovvero quando abbiamo entropia della sorgente pari a $8\,\mathrm{entropy\ bit/Byte}$. Nel caso reale, il TRNG di nostra concezione ha mostrato entropia pari a $7.98\,\mathrm{bit/Byte}$, approssimativamente coincidente col caso ideale. L'`entropy_count` viene aggiornato di conseguenza.

Una prima verifica sulla qualità del distiller di Linux™, sebbene approssimativa, consiste nell'osservare la statistica prodotta in uscita dal LRNG quando questo utilizza il TRNG come sorgente di entropia. Come mostrato in Fig. 5.13, la sequen-

za di valori ottenuti dall'interfaccia `dev/random` ha una statistica con andamento uniforme, paragonabile a rumore bianco (vedi immagine sulla destra). Nei test effettuati, il massimo data rate in lettura da `dev/random` è stato $32\,\mathrm{Kbit/s}$ su una sequenza binaria di $100 \times 10^6\,\mathrm{bit}$. È bene notare come un tale throughput continuativo sull'interfaccia bloccante sia garantito solo dalla presenza del TRNG oggetto della proposta di questa tesi. In caso contrario, `dev/random` si bloccherebbe dopo qualche centinaio di bit. Inoltre, il modesto data rate ottenuto ha permesso di effettuare test statistici accurati (vedi Cap.6) in tempi ridotti, operazione impensabile altrimenti.

Capitolo 6

Validazione mediante test

Come in tutti i generatori HRNG, la generazione di numeri casuali è affidata a sorgenti di rumore progettate su hardware elettronico. Quindi, uno dei primi controlli da attuare sussiste nel verificare che la sorgente di rumore rimanga entro i parametri di funzionamento nominali. A questo scopo, possono essere utilizzati test statistici, i quali spesso permettono di individuare il guasto della sorgente, prima che questo possa compromettere la qualità delle sequenze generate [50] [47]. Sebbene, piccole deviazioni dal modello nominale del generatore siano comunque indice di un corretto funzionamento, deviazioni più importanti possono essere sintomo di un degrado della sorgente di rumore.

L'uscita stessa della sorgente rumorosa dovrebbe essere testata prima di attraversare un blocco di *whitening*[12], in quanto alcune funzioni di rimescolamento complesse, presenti al suo interno, possono portare a passare i test anche in condizioni di ingressi non casuali.

Sfortunatamente, i test attualmente disponibili (e probabilmente quelli che lo saranno in futuro) non sono in grado di identificare l'impredicibilità di una sorgente casuale. Passare tutti i test non è sufficiente per affermare che le sequenze casuali prodotte siano realmente *random*. Infatti, qualsiasi tipo di test possa essere condotto, valuta solamente la qualità statistica di un numero finito di sequenze. Le informazioni limitate così ottenute, costituiscono unicamente un sottoinsieme di tutti i possibili risultati. Per tale motivo un test non sarà mai in grado di confermare

[12]Effettua una de-correlazione sulla sequenza in ingresso, producendo una sequenza di uscita paragonabile a rumore bianco (white noise).

in modo assoluto l'impredicibilità di un generatore, sia esso un PRNG che un TRNG.

Addirittura, da questo punto di vista, potremmo ottenere risultati migliori nei test condotti su PRNG, rispetto a quelli portati a termine su TRNG. Questo non deve sorprendere, in quanto, similarmente ai PRNG, anche l'algoritmo di test è una macchina a stati finiti: conoscere come questa sia implementata, ad esempio, può fornire le basi per progettare un PRNG *ad hoc* in grado di passare agevolmente il test. Inoltre, maggiore è la complessità del test, più difficile diventa passarlo per un TRNG, se confrontato con un PRNG, in quanto sarà sempre possibile costruire un PRNG sufficientemente complesso per riuscire a produrre le esatte statistiche richieste. Consideriamo, per esempio, un test che valuta correlazioni fino a 20 passi. Se volessimo costruire un PRNG in grado di passare il test, sarebbe sufficiente progettarlo in modo tale che eventuali correlazioni siano spalmate su intervalli maggiori di 20 passi.

6.1 I test FIPS 140-2 e NIST 800-22

I TRNG dovrebbero essere costantemente monitorati per verificarne il comportamento nominale. A tale scopo, possono essere condotti svariati test, atti a provare che le proprietà statistiche delle sequenze generate siano quelle attese. Raccomandazioni di sicurezza e standard di implementazione dei suddetti test sono riportati nei documenti RFC 4086 [19], FIPS 140-2 [38], NIST 800-22rev1a [39] e NIST 800-90b [40]. Di seguito, descriviamo brevemente caratteristiche e funzionamento del FIPS 140-2 e NIST 800-22.

Gli standard serie 140 definiti dalla Federal Information Processing Standards (FIPS) specificano i requisiti per le unità crittografiche, senza però fornire condizioni sufficienti a garantire che un modulo conforme a tali requisiti sia sicuro (tantomeno un sistema che impiega tale modulo). I componenti della serie 140 sono, ordinati per anno di pubblicazione, il FIPS 140-1 (1994), il FIPS 140-2 (2001) e il FIPS 140-3 (ancora in fase di sviluppo). Questi standard impongono requisiti su undici aree differenti, tra le quali compare quella di *self-test*. Generatori garantiti secondo gli standard FIPS 140 devono essere dotati di un algoritmo in grado di verificarne periodicamente, ad esempio ogni $20000\,\mathrm{bit}$ generati, il funzionamento nominale. A questo scopo, possono essere condotti i seguenti test statistici:

- **Monobit test**: conta le occorrenze X di `1` in una sequenza di $20000\,\mathrm{bit}$ generati. Il risultato del test è positivo se $9654 < X < 10346$.

- **Poker test**: divide la sequenza di $20000\,\mathrm{bit}$ in 5000 segmenti di $4\,\mathrm{bit}$, conteggiando le occorrenze $f(i)$ delle 16 possibili combinazioni di $4\,\mathrm{bit}$, dove $i = 0, 1, ..., 15$. Il risultato del test è positivo se $1.03 < X < 57.4$, dove

$$X = \frac{16}{5000}\left(\sum_{i=0}^{15} f(i)^2\right) - 5000$$

- **Runs test**: una corsa è definita come la massima ripetizione di `0` o `1` consecutivi, rilevati in una sequenza di $20000\,\mathrm{bit}$. Vengono contate le occorrenze di tutte le corse (sia `0` che `1`) di tutte le lunghezze (≥ 1). Il risultato del test è positivo se tutte le lunghezze delle corse rimangono all'interno degli intervalli mostrati in tabella:

Lunghezza corsa	Occorrenze minime	Occorrenze massime
1	2267	2733
2	1079	1421
3	502	748
4	223	402
5	90	223
6+	90	223

- **Long Run test**: una corsa lunga è definita tale quando la lunghezza è ≥ 34. Il risultato del test è positivo quando, su una sequenza di $20000\,\mathrm{bit}$, non si verificano corse lunghe.

Batterie di test di questo tipo sono agevolmente implementabili nel codice del generatore TRNG e consentono di identificare nel corso della generazione il verificarsi di situazioni anomale. Tali condizioni possono essere legate, ad esempio, a interferenti esterni e comportano l'arresto del TRNG. Naturalmente, considerata la semplicità della loro struttura, test simili non hanno come scopo quello di verificare le proprietà statistiche delle sequenze prodotte. Quasi tutti i TRNG in commercio implementano il self test riportato nel FIPS 140-2. Lo stesso test potrebbe essere implementato sul TRNG oggetto di questa tesi.

Raccomandazioni più recenti sono quelle contenute nei documenti 800-22rev1a e 800-90b prodotti dal National Institute of Standard and Technology (NIST) [39] [40], sui quali è progettato l'applicativo Statistical Test Suite (STS) 2.1 rilasciato dallo stesso NIST. Il tool STS 2.1 è composto da 15 test, chiamati *Frequency* (FR), *Block Frequency* (BF), *Cumulative Sums* (CS), *Runs* (RU), *Longest Runs of Ones* (LR), *Rank* (MR), *Spectral - Discrete Fourier Transform* (SP), *Nonperiodic Template Matchings* (NOT), *Overlapping Template Matchings* (OT), *Universal Statistical* (UN), *Approxiamate Entropy* (AE), *Random Exursions* (RE), *Random Excursions Variant* (REV), *Serial* (SE) e *Linear Complexity* (LC). Ognuno dei test esamina un differente comportamento statistico, producendo in uscita un indicatore statistico, chiamato *P-value*. Il P-value è un numero reale nell'intervallo $[0, 1]$ e fornisce una stima della probabilità che una realizzazione finita di un processo binario casuale ideale diverga dalla statistica ideale più di quanto faccia la sequenza testata. Quindi, maggiore è il P-value, più casuale è la sequenza. Diremo che il test è passato se il P-value è maggiore di α'.

Siccome il test viene condotto sui soli dati a disposizione, il P-value stesso è da considerare come una variabile aleatoria, testando un numero sufficiente di sequenze. La distribuzione che otterremo, nel caso di un generatore di bit IID, sarebbe uniforme in $[0, 1]$. Perciò, in media, il rapporto tra il numero di sequenze che passano il test e quelle testate è $1 - \alpha'$.

Considerato che il numero di sequenze considerate è finito, è normale aspettarsi qualche scostamento da tale proporzione. A tale scopo, le raccomandazioni NIST

suggeriscono l'adozione del criterio 3σ. Quindi, diremo che il numero di sequenze che passano il test è compatibile con la casualità della sorgente solo se quest'ultimo ricade nell'intervallo

$$1 - \alpha' \pm 3\sqrt{\alpha'(1-\alpha')/T} \tag{6.1}$$

dove T è il numero di sequenze testate.

Ultimata la fase di test, il tool STS 2.1 produce in uscita il file `finalAnalisysReport.txt`, nel quale è riportato un riepilogo dei risultati ottenuti. Questi ultimi sono presentati in forma di tabella, dove il numero di righe p corrisponde al numero di test statistici condotti, mentre il numero di colonne q è fissato. Queste ultime sono così suddivise:

- $q = 1 \div 10$: frequenza dei P-value, la quale può essere visivamente illustrata mediante un istogramma, come in Fig. 6.1. L'intervallo unitario $[0, 1]$ è stato diviso in dieci sottointervalli e i P-value che si posizionano all'interno di essi sono conteggiati.

- $q = 11$: P-value ottenuto dall'applicazione del χ^2 test[13].

- $q = 12$: proporzione delle sequenze binarie che hanno passato il test.

- $q = 13$: nomenclatura del generico test condotto.

Nello stesso file, in fondo, è riportata la minima proporzione tra test passati e test condotti per poter considerare il generico test passato. Per facilitare ulteriormente la comprensione dei risultati, un asterisco riportato a fianco del valore può indicare la presenza di un test non passato (proporzione al di sotto del valore minimo) o una distribuzione dei P-value non uniforme (risultato del χ^2 test inferiore a 10^{-4}).

Batterie di test di questo tipo sono molto più complesse rispetto a quelle precedenti e pretendono capacità computazionali e di storage elevate. Non implementabili su sistemi a μC o sistemi embedded, test statistici simili vengono condotti lato client (su computer). Tali verifiche intensive vengono effettuate in fase di sviluppo del TRNG e permettono di identificare architetture di generatori

[13] Usato per valutare l'uniformità nella distribuzione dei P-value nel generico test. Valori superiori a 10^{-4} sono compatibili a sorgenti casuali perfette. Inoltre, per ottenere risultati statisticamente validi, almeno 55 sequenze devono essere processate.

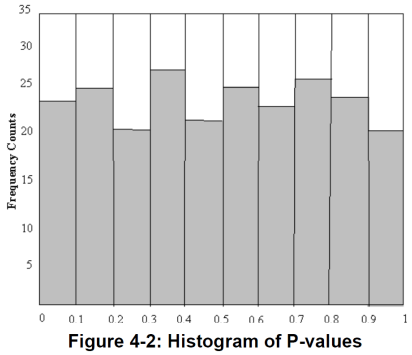


Figura 6.1: Esempio di istogramma riportante una distribuzione uniforme dei P-value. (fonte [39])

fuori specifica. Nella prossima sezione, condurremo svariati test statistici sulle sequenze prodotte dal TRNG oggetto della proposta di tesi, in diverse condizioni di lavoro, sfruttando il tool NIST STS 2.1.

6.2 Connessione del TRNG ad un sistema Linux™

Come già anticipato nelle Sez. 5.5.1 e 5.5.2 del Cap. 5, il SO Linux™ dispone di un'architettura per la raccolta di entropia e la generazione di numeri casuali.

Focalizziamo ora l'attenzione sulla fase di immagazzinamento dell'entropia nell'entropy pool primario. Per poter conoscere una stima dell'entropia contenuta nell'entropy pool primario è sufficiente leggere il valore riportato nel file `/proc-/sys/kernel/random/entropy_avail`, dove il valore massimo raggiungibile è `/proc/sys/kernel/random/poolsize`(4096 bit nell'ultima versione del kernel). Un applicativo, compilato in Processing© [11], mostra visivamente la dinamica di questi due valori nel tempo, come illustrato in Fig. 6.2.

Tale visualizzazione permette di avere un riscontro immediato sullo stato dell'entropy pool primario. Se le sole sorgenti di entropia sono mouse, tastiera, latenze HDD e interrupt di sistema, il livello di entropia evidenziato è ridotto e aumenta lentamente. Al contrario, impiegando il TRNG come sorgente di entropia, il livello di entropia si assesta al valore massimo e, anche prelevando dagli entropy pool secondari, lì rimane. Quest'ultimo caso è evidenziato in Fig. 6.3.

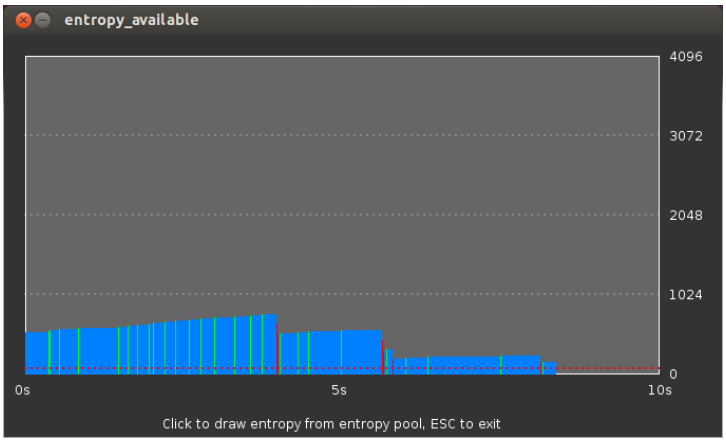


Figura 6.2: Script Processing©per la visualizzazione della stima dell'entropia contenuta nell'entropy pool primario. Sorgenti di entropia disponibili al kernel: mouse, tastiera, latenze HDD e interrupt di sistema.

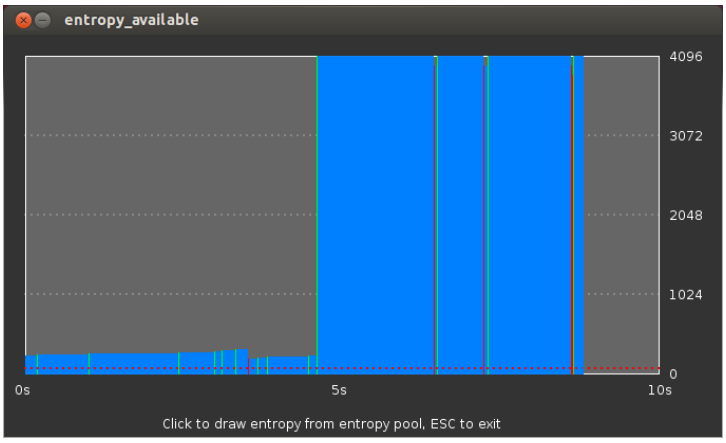


Figura 6.3: Script Processing©per la visualizzazione della stima dell'entropia contenuta nell'entropy pool primario. Sorgenti di entropia disponibili al kernel: TRNG, mouse, tastiera, latenze HDD e interrupt di sistema. Il TRNG viene avviato all'istante di tempo $t = 4.7\,\mathrm{s}$.

Nella Sez. 5.4 abbiamo evidenziato la metodologia con cui trasferire i valori casuali prodotti dal TRNG all'entropy pool primario. Tale processo avviene sfruttando la comunicazione seriale USB CDC. Connettendo il TRNG ad un PC con SO Linux™, questo viene rilevato come un emulatore seriale USB. Possiamo verificare la connettività e recuperare l'interfaccia virtuale tramite il comando

```
sudo dmesg | grep 'USB ACM'
[73084.365651] cdc_acm 2-2.1:1.0: ttyACM0: USB ACM device
```

dove `ttyACM0` è l'identificativo del TRNG.

Appurato che il TRNG sia raggiungibile all'interfaccia `dev/ttyACM0`, possiamo avviare il demone studiato nella Sez. 5.5.2, il quale saturerà istantaneamente l'entropy pool primario. Il processo di *feeding* continuerà fino a quando il demone sarà attivo. A questo punto possiamo prelevare con continuità dall'interfaccia bloccante `dev/random`, tramite il comando

```
dd if=/dev/random of=./random.txt bs=1 count=12500000
12500000+0 records in
12500000+0 records out
12500000 bytes (12.5 MB) copied, 5952.489 s, 2.1 kB/s
```

In questo modo i valori casuali letti dal `dev/random` vengono memorizzati sul file `random.txt`. Allo stesso tempo vengono memorizzati, su un file separato, i valori grezzi ricevuti dal TRNG. Sarà così possibile, in seguito, grazie al tool NIST STS 2.1, valutare le proprietà statistiche dei due flussi distinti. I risultati attesi nei due casi sono:

- in ingresso all'entropy distiller di Linux™: i dati grezzi ricevuti dal TRNG possiedono già buone caratteristiche statistiche (secondo la teoria vista nel Cap. 4);
- in uscita dall'entropy distiller di Linux™: i bit casuali letti da `dev/random` sono IID (vista la complessità dell'architettura dello stesso, riportata nella Sez. 5.5.1).

La quantità di dati memorizzati è pari a $12500000\,\mathrm{byte} \equiv 10^8\,\mathrm{bit}$ per ogni nuova configurazione del sistema. Tale valore è il minimo necessario per condurre

i test su 100 sequenze, ognuna di lunghezza $10^6\,\mathrm{bit}$, come descritto nella prossima sezione. Inoltre, è doveroso notare il throughput medio raggiunto[14] in lettura da `dev/random`, pari a $2.1\,\mathrm{kB/s} \equiv 16.8\,\mathrm{kbps}$. Velocità simili in tale contesto sarebbero impensabili in assenza della sorgente TRNG. Tuttavia, il data rate ottenuto non è ancora comparabile con le altre soluzioni commerciali. Ciononostante, tale limite può essere superato attuando un'ottimizzazione del codice (argomento al di fuori della portata di questa trattazione).

6.3 Validazione statistica del TRNG

Per validare il TRNG in diverse condizioni di lavoro abbiamo voluto considerare otto configurazioni distinte:

- caso A: il TRNG realizza la mappa M_{ADC} a quattro rami, come mostrato in Fig. 5.8. I due bit prodotti ad ogni iterazione della mappa vengono memorizzati e poi inviati senza effettuare alcun post-processing. Lo schema di codifica impiegato è 00-01-10-11-00. La sequenza analizzata è prelevata in ingresso all'entropy distiller di Linux™.

- caso B: la sequenza analizzata è prelevata in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso A.

- caso C: il TRNG realizza la mappa M_{ADC} a quattro rami, come mostrato in Fig. 5.8. I due bit prodotti ad ogni iterazione della mappa vengono memorizzati e poi inviati senza effettuare alcun post-processing. Differentemente dal caso A, in questo caso viene impiegato lo schema di codifica alternativo 01-00-10-11-01. La sequenza analizzata è prelevata in ingresso all'entropy distiller di Linux™.

- caso D: la sequenza analizzata è prelevata in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso C.

[14] Ottenuto applicando post-processing in uscita dal TRNG, il quale comporta una riduzione pari a un quarto del data rate.

- caso E: il TRNG realizza la mappa M_{ADC} a quattro rami, come mostrato in Fig. 5.8. I due bit prodotti ad ogni iterazione della mappa vengono memorizzati e successivamente processati per mezzo dell'algoritmo Von Neumann, mostrato in Tab. 3.1. Lo schema di codifica impiegato è quello alternativo, 01-00-10-11-01. La sequenza analizzata è prelevata in ingresso all'entropy distiller di Linux™.

- caso F: la sequenza analizzata è prelevata in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso E.

- caso G: il TRNG è disconnesso dal sistema perché inutilizzato in questa configurazione. La sequenza analizzata, prelevata in ingresso all'entropy distiller di Linux™, è deterministica. Essa viene generata mediante uno script, replicando iterativamente il pattern binario `01`.

- caso H: la sequenza analizzata è prelevata in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso G.

Le otto configurazioni previste forniscono un ottimo punto di partenza per valutare le prestazioni del TRNG sviluppato. Allo scopo di ridurre il numero di combinazioni possibili, si è utilizzata unicamente la mappa M_{ADC} a 4 rami, visibile in Fig. 5.8. I parametri utilizzati sono gli stessi impiegati nella Sez. 5.4.1. Durante la sperimentazione del TRNG sono state testate efficacemente anche mappe a più rami (8, 16 e 32 rami), grazie alle quali è stato possibile ottenere un numero maggiore di bit generati per ogni iterazione, incrementando così il throughput del TRNG. La scelta di riportare unicamente i risultati ottenuti con la mappa M_{ADC} a 4 rami ha il solo scopo di semplificare la trattazione.

Inoltre, in ognuna delle situazioni evidenziate si è cercato di minimizzare il contributo dovuto alle sorgenti di entropia del kernel Linux™. A tale scopo, mouse e tastiera sono stati disattivati e gli accessi alle risorse di sistema (come la scrittura o lettura sull'HDD) sono stati limitati.

Perché l'applicativo STS 2.1 possa produrre risultati statisticamente validi, scegliamo di condurre i test su $T = 100$ sequenze differenti, ognuna lunga 10^6 bit. Questo presuppone una lunghezza minima della sequenza di bit casuali prodotta dal TRNG pari a $100 \times 10^6\,\text{bit} = 10^8\,\text{bit}$. Inoltre, la soglia minima selezionata

Test	A	C	E	XOR	G
FR	0/100	0/100	99/100	98/100	100/100*
BF	0/100	81/100*	98/100	99/100	100/100*
CS	0/100	0/100	100/100	98/100	100/100*
RU	0/100	0/100	56/100*	100/100	0/100
LR	100/100	99/100	99/100	97/100	0/100
MR	99/100	100/100	98/100	98/100	0/100
SP	98/100	99/100	98/100	96/100	0/100
NOT	0/100	59/100*	95/100	95/100	0/100
OT	36/100*	98/100	97/100	97/100	0/100
UN	57/100*	96/100	99/100	99/100	0/100
AE	0/100	70/100*	98/100	98/100	0/100
RE	1/1*	4/4*	60/62	41/42	-
REV	1/1*	4/4*	59/62	41/42	-
SE	0/100	96/100	98/100	99/100	0/100
LC	100/100	100/100	97/100	99/100	0/100

Tabella 6.1: Tabella di comparazione dei risultati ottenuti tramite tool NIST STS 2.1 nelle diverse condizioni: assenza di post-processing e schema di codifica 00-01-10-11-00 (A), assenza di post-processing e schema di codifica alternativo 01-00-10-11-01 (C), applicazione dell'algoritmo di post-processing di Von Neumann (E), sequenza ottenuta mediante post-processing XOR ($4\,\mathrm{bit/blocco}$) (XOR), sequenza deterministica con pattern `01` (G).

per i P-value è $\alpha' = 0.01$, da cui deriva un *pass rate* pari a $1 - \alpha' = 0.99$. Tuttavia, se consideriamo il criterio 3σ, già citato in precedenza, otteniamo l'intervallo di validità dei test $[0.96, 1]$, al quale corrisponde una proporzione minima di $96/100$. Quest'ultimo valore vale per tutti i test presenti nel tool STS 2.1, ad eccezione di *Random Excursions* (RE) e *Random Excursions Variant* (REV) per i quali si hanno le proporzioni minime di $39/42$, $57/60$, $58/61$ e $60/64$. Nel caso di risultati multipli attinenti lo stesso test, il risultato riportato nelle tabelle a seguire è quello peggiore.

In Tab. 6.1 sono riportate le proporzioni (test passati su test condotti) nelle quattro configurazioni A, C, E e G, nelle quali i dati analizzati sono stati prelevati in ingresso all'entropy distiller di Linux™. I risultati completi ottenuti grazie al tool NIST STS 2.1 sono riportati in Appendice nelle Sez. A.4, A.6, A.8 e A.10.

Sulla base dei risultati statistici riportati in Tab. 6.1 possiamo fare le seguenti considerazioni

- caso G: la sequenza deterministica () non passa nessuno dei test, come da aspettativa;
- caso A: la sequenza generata dal TRNG in assenza di post-processing passa solo qualche test;
- caso C: il risultato precedente migliora di poco quando adottiamo lo schema di codifica alternativo 01-00-10-11-01;
- caso E: l'applicazione dell'algoritmo di Von Neumann produce grossi benefici in termini statistici.

Grazie all'algoritmo di Von Neumann, abbiamo ottenuto una sequenza binaria in grado di passare tutti i test NIST, ad eccezione del *Runs test* (RU). Tuttavia, la sequenza così elaborata mostra un datarate non più costante.

Per ovviare all'esito negativo nel test NIST e normalizzare il datarate della sequenza binaria casuale, è stato necessario adottare un post-processing differente: i bit generati dal TRNG sono raggruppati in blocchi ($4\,\mathrm{bit/blocco}$ nel nostro caso) e l'uscita binaria equivale all'operazione di XOR tra i bit appartenenti allo stesso blocco. In questo modo, si ottiene sia un datarate costante di generazione, pari a un quarto di quello originale, sia qualità statistiche migliori. Infatti, come mostrato nella colonna XOR in Tab. 6.1, tale metodo permette di passare brillantemente tutti i test NIST. I risultati completi ottenuti grazie al tool NIST STS 2.1 sono riportati in Appendice nella Sez. A.12.

Per verificare la qualità complessiva dei numeri casuali in un sistema Linux™ abbiamo condotto i test NIST anche sulle sequenze casuali prelevate da `dev/random`, sotto differenti condizioni di lavoro. In Tab. 6.2 sono riportate le proporzioni (test passati su test condotti) nelle quattro configurazioni B, D, F e H, nelle quali i dati analizzati sono stati prelevati in uscita dall'entropy distiller di Linux™, in presenza delle sequenze A, C, E e G in ingresso. I risultati completi ottenuti grazie al tool NIST STS 2.1 sono riportati in Appendice nelle Sez. A.5, A.7, A.9 e A.11.

Sulla base dei risultati statistici riportati in Tab. 6.2 possiamo concludere che il LRNG ha un comportamento ideale, qualsiasi sia la sequenza fornita al suo ingresso. La complessità della sua architettura garantisce la generazione di numeri casuali

Test	B	D	F	H
FR	100/100	100/100	100/100	100/100
BF	97/100	99/100	100/100	100/100
CS	100/100	100/100	100/100	99/100
RU	99/100	99/100	100/100	98/100
LR	99/100	98/100	99/100	99/100
MR	99/100	99/100	100/100	99/100
SP	99/100	97/100	97/100	98/100
NOT	96/100	94/100	96/100	95/100
OT	99/100	99/100	99/100	97/100
UN	98/100	98/100	99/100	99/100
AE	99/100	99/100	99/100	99/100
RE	57/60	62/64	63/64	60/61
REV	57/60	62/64	62/64	59/61
SE	97/100	99/100	99/100	99/100
LC	99/100	98/100	100/100	99/100

Tabella 6.2: Tabella di comparazione dei risultati ottenuti tramite applicazione del tool NIST STS 2.1 sulle sequenze di uscita dell'entropy distiller di Linux™, in presenza delle sequenze A, C, E e G in ingresso.

perfetti anche in condizioni proibitive, come quelle in cui in ingresso vengono passate sequenze deterministiche (caso H), ad eccezione di qualche imperfezione marginale (vedi il risultato peggiore ottenuto nel test NOT).

Data la natura dei test effettuati, i risultati ottenuti devono essere interpretati statisticamente. Quindi, con particolare riferimento al caso in cui viene applicato un post-processing di tipo XOR, quello che possiamo concludere è che, anche in presenza di non idealità nell'implementazione della mappa M_{ADC} , il TRNG oggetto della proposta di questa tesi

- conserva ancora un comportamento caotico con una probabilità indistinguibile da 1;
- preserva la propria capacità di generare sequenze binarie che passano il test NIST con una probabilità indistinguibile da 1;
- garantisce la generazione di sequenze *random*, con caratteristiche comparabili a quelle in uscita dall'entropy distiller di Linux™, con una probabilità indistinguibile da 1.

6.4 Stima dell'entropia del TRNG

Un metodo alternativo, anche più corretto dal punto di vista teorico, per valutare la casualità di un determinato processo è la stima della sua entropia. Sebbene vi siano svariati metodi per misurare tale entropia, quella che utilizzeremo di seguito è la definizione di Shannon [53]

$$H \triangleq \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \triangleq -\sum_{i=1}^{n} p_i \log_2 p_i \tag{6.2}$$

dove i termini p_i rappresentano la probabilità che il processo sotto esame generi l'uscita i-esima date n possibili uscite. In questo caso la precedente formula è misurata in bit, in quanto compare il logaritmo in base due. L'entropia di Shannon, così definita, misura la quantità media di informazione, espressa in bit, contenuta nella sequenza di uscita del processo. Se un RNG genera in uscita vettori binari di k bit, p_i è la probabilità che l'uscita sia uguale a i, dove $0 \leq i \leq 2^k$. Quindi, nel caso di RNG perfetti tale probabilità vale $p_i = \frac{1}{2^k} = 2^{-k}$ e l'entropia dell'uscita del processo è pari a $k\,\mathrm{bit}$. Questo comporta che tutte le possibili uscite siano equiprobabili e che, in media, l'informazione presente in uscita non può essere rappresentata in un sequenza con lunghezza inferiore a $k\,\mathrm{bit}$. Ovviamente, avvicinare tale limite è l'obiettivo di un buon TRNG.

Anche se esistono stimatori di entropia più sofisticati (in confronto a quelli che si basano sulla definizione stessa), per l'applicazione proposta, lo stimatore basato sull'espressione 6.2 è particolarmente adeguato. Infatti la dinamica caotica sottostante garantisce che tutte le correlazioni siano evanescenti con una decrescita esponenziale all'aumentare del riferimento temporale. Questo significa che anche restringendo la somma in 6.2 a valori di n relativamente piccoli, per i processi presi in considerazione si può avere una buona stima dell'entropia.

Adottando la formulazione 6.2, abbiamo misurato l'entropia di Shannon delle sequenze binarie generate dal TRNG nelle differenti configurazioni testate in precedenza, riportando i risultati ottenuti in Tab. 6.3. La misurazione è stata portata a termine dividendo, volta per volta, la sequenza binaria in blocchi di $m\,\mathrm{bit}$, dove $1 \leq m \leq 16$ (valore massimo $m = 16$ scelto in accordo con le capacità computazionali a disposizione). Ognuna delle righe riporta il valore normalizzato

m	A	C	E	XOR
1	0.99994556	0.99996438	0.99999999	1.00000000
2	0.99968746	0.99995436	0.99999733	0.99999996
3	0.99962449	0.99995166	0.99999633	0.99999986
4	0.99871027	0.99994932	0.99999591	0.99999972
5	0.99881834	0.99994843	0.99999553	0.99999957
6	0.99717896	0.99994763	0.99999517	0.99999896
7	0.99831449	0.99994684	0.99999487	0.99999829
8	0.99571646	0.99994613	0.99999307	0.99999605
9	0.99776474	0.99994508	0.99999269	0.99999310
10	0.99412775	0.99994277	0.99998976	0.99998587
11	0.99723060	0.99993907	0.99998466	0.99996862
12	0.99256246	0.99993137	0.99997556	0.99993853
13	0.99671813	0.99991559	0.99995500	0.99987800
14	0.99176927	0.99988568	0.99991594	0.99976241
15	0.99614582	0.99982726	0.99983625	0.99952092
16	0.99018249	0.99970670	0.99967694	0.99904039

Tabella 6.3: Tabella di comparazione dell'entropia di Shannon misurata sulle sequenze binarie generate dal TRNG nelle configurazioni A, C, E e post-processing di tipo XOR. La sequenza binaria è divisa in blocchi di m bit, dove $1 \leq m \leq 16$ (valore massimo $m = 16$ scelto in accordo con le capacità computazionali a disposizione). Ognuna delle righe riporta il valore normalizzato rispetto a m.

rispetto a m. In Tab. 6.3 sono riportati i risultati ottenuti nelle configurazioni in cui la generazione avviene per mezzo del TRNG:

- caso A: anche in assenza di post-processing i valori di entropia ottenuti sono soddisfacenti;
- caso C: ancora in assenza di post-processing, adottare lo schema di codifica 01-00-10-11-01 aiuta ad incrementare, anche se di poco, l'entropia della sequenza prodotta;
- caso E: elaborando la sequenza generata nella configurazione C con l'algoritmo di Von Neumann otteniamo valori di entropia prossimi a 1.

Siccome le stime di entropia di Shannon ottenute sono tutte prossime all'unità, possiamo dire che, in media, il TRNG oggetto della proposta di questa tesi ben

approssima il comportamento di un RNG perfetto, in grado di generare sequenze *autenticamente casuali*.

Capitolo 7

Conclusione

In questo elaborato è stata presentata un'architettura originale per la generazione di numeri autenticamente casuali.

L'architettura soddisfa i requisiti per l'applicazione in sistemi che richiedono sequenze casuali anche per compiti strategici, come la crittografia e la sicurezza informatica.

Inoltre, l'architettura è stata progettata in modo tale da renderla fruibile su sistemi pre-esistenti che non incorporano generatori di numeri autenticamente casuali. Con ciò, il sistema proposto ha la potenzialità di aumentare la sicurezza informatica su *sistemi embedded* e di adeguare a nuovi parametri di sicurezza alcune architetture *legacy*.

Aspetti salienti dell'architettura proposta sono il bassissimo costo implementativo (si stima che su medi volumi l'intera architettura possa venire prodotta con meno di 20$[15]) e la versatilità applicativa. Quest'ultima è garantita dal fatto che l'intera piattaforma è costruita attorno a un microcontrollore che, sebbene economico, mette a disposizione svariate opzioni di comunicazione, tra cui il bus USB (diventato oramai un *must* su dispositivi elettronici). A titolo di riferimento, applicazioni concorrenti sul mercato hanno prezzi di vendita unitari che oscillano tra 37$ e 150$, come evidenziato precedentemente in Tab. 3.2. Inoltre, il risultato ottenuto, in un'ottica di *low cost*, *low power* e *low size*, è importante nel contesto di Internet of Things, nel quale è fondamentale l'interconnessione in sicurezza di

[15] Al costo dei soli componenti nella BOM, mostrata in Tab. 5.6, vanno aggiunti i costi di produzione della PCB e di montaggio.

un numero enorme di dispositivi elettronici economici a basso impatto ambientale.

Benché basata su microcontrollore, l'architettura sfrutta una catena di signal processing analogica per la generazione dei valori casuali. Questa caratteristica è di fondamentale importanza, in quanto su sistemi perfettamente digitali, riconducibili a macchine a stati finiti, la sintesi di numeri autenticamente casuali è impossibile. Inoltre, è doveroso notare come la sezione fondamentale della catena di elaborazione analogica sia costituita da un convertitore A/D pre-integrato nel microcontrollore. Ciò consente di utilizzare un blocco funzionale su cui si hanno buone garanzie di accuratezza e, al tempo stesso, ottenere l'abbattimento dei costi.

La possibilità di riutilizzare l'ADC è resa possibile dallo sfruttamento di un meccanismo di generazione basato su una dinamica caotica. Questo approccio non è comune tra i generatori autenticamente casuali attualmente presenti sul mercato. Tuttavia, offre di per sé alcuni vantaggi, il principale dei quali risiede nella possibilità di dimostrare formalmente (con un approccio matematico) che l'architettura è in grado di generare sequenze perfettamente casuali in condizioni ideali. Al contrario, adottando metodi convenzionali, anche in condizioni ideali, l'unica verifica possibile sulla qualità delle sequenze generate rimane quella sperimentale.

In ogni caso, sono stati condotti anche svariati test sperimentali, a validazione di quanto appena detto. A tale scopo, si è fatto riferimento a stime di entropia e all'uso di batterie di test statistici standardizzate (NIST 800-22). Queste ultime sono state applicate sia al generatore puro, sia al generatore coadiuvato da algoritmi di post-processing, sia al generatore utilizzato come sorgente nella sintesi di numeri casuali in un sistema operativo Linux™. Già nel primo caso sono state evidenziate caratteristiche statistiche soddisfacenti, sebbene imperfette. Infine, negli ultimi due casi le prestazioni ottenute sono ideali, consentendo di classificare il generatore come *cryptographically secure*.

7.1 Possibili Sviluppi

Considerata la portata dell'elaborato, il proseguimento delle attività in termini di pre-industrializzazione è tra i possibili scenari futuri. Infatti è già disponibile un prototipo tangibile ed è facile immaginare un'ottimizzazione del *dongle* USB

TRNG oggetto della proposta di tesi. I nuovi vincoli di progetto sarebbero ancora più stringenti in merito a robustezza, throughput e costo. A tale scopo, potrebbe essere sostituito il costoso μC PIC18F2550, rimpiazzandolo con il più economico PIC18F14K50. Tale modifica potrebbe garantire la fruizione del DAC interno al μC, in sostituzione a quello esterno attuale. Inoltre, come già anticipato nella Sez. 6.1, sarebbe fondamentale integrare algoritmi di *self test*, come il FIPS 140-2, nel firmware del dispositivo. In tal modo si potrebbe consentire al sistema di rilevare autonomamente eventuali fenomeni interferenti esterni o guasti interni in modo da garantire le proprietà statistiche della sequenza binaria generata.

Non meno importante, invece, l'aspetto della sicurezza nella comunicazione tra TRNG e sistema *host*. Infatti, è nota la possibilità di utilizzare *sniffer* USB allo scopo di carpire i messaggi scambiati sul bus. A tutt'ora non è stato implementato alcun meccanismo di sicurezza della comunicazione tra TRNG e *host*. Tale aspetto è maggiormente percepito nei sistemi che adottano un bus USB condiviso. Conseguentemente, come soluzione temporanea, in assenza di un opportuno protocollo di comunicazione sicura, è opportuno usare un bus USB dedicato o renderlo difficilmente accessibile.

Come ultimo punto, non di secondaria importanza, è quello relativo ai *side channel attacks*. Noti in crittografia, permettono di estrapolare dati sensibili derivandoli da informazioni ottenute dall'implementazione fisica del sistema crittografico. Per esempio, un possibile *side channel* della nostra architettura potrebbe essere il consumo di potenza. Acquisizioni successive della corrente assorbita dall'alimentazione potrebbero fornire informazioni relative alla variabile di stato del sistema caotico, rendendo possibile predire quale sarà la sequenza casuale generata. Ci si può aspettare che a porre rimedio a questo inconveniente sia la natura stessa del sistema caotico: anche conoscendo lo stato interno del sistema in un determinato momento, data la velocità con cui decadono le correlazioni in un sistema caotico (vedi Cap. 4), sarebbe impossibile a chiunque, anche conoscendo l'architettura dello stesso, predire stati futuri al di là dell'orizzonte temporale. Ciononostante, una verifica sperimentale sarebbe opportuna.

Elenco delle figure

Elenco delle tabelle

Appendice A

Appendice

Microchip Advanced Part Selector

Microcontroller Side-by-Side Compare

Part	**PIC18F14K50**	**PIC18F24K50**	**PIC18F25K50**	**PIC18F2550**
Manufacturer	Microchip	Microchip	Microchip	Microchip
P.Memory (Kbytes)	16 Flash	16 Flash	32 Flash	32 Flash
P.Memory (KWords)	8	8	16	16
Self-Write Flash	Yes	Yes	Yes	Yes
RAM(Bytes) EEPROM(Bytes)	768 256	2K 256	2K 256	2K 256
I/O Pins	15	25	25	24
Max CPU Speed / Internal OSC	48 MHz (12 MIPS) 16 MHz, 32 kHz	48 MHz (12 MIPS) 16 MHz, 32 kHz	48 MHz (12 MIPS) 16 MHz	48 MHz (12 MIPS) 8 MHz, 32 kHz
System Mgmt Features	Programmable BOR, None, POR, WDT, nanoWatt-[Low Speed, Fast Wake, Active Ctrl]	Programmable BOR, Programmable LVD, WDT, nanoWatt-[Low Speed, Fast Wake, Active Ctrl]	Programmable BOR, Programmable LVD, WDT, nanoWatt-[Low Speed, Fast Wake, Active Ctrl]	Programmable BOR, Programmable LVD, POR, WDT, nanoWatt-[Low Speed, Fast Wake, Active Ctrl]
Analog Peripherals	2-Comparators w/SR-Latch, Bandgap - Yes; 1A/D, 9x10-bit @ 100ksps; 0D/A, 0x0-bit @ ksps	2-Comparators, Bandgap - Yes; 1A/D, 14x10-bit @ 100ksps; 1D/A, 1x5-bit @ 100ksps, CTMU	2-Comparators, Bandgap - Yes; 1A/D, 14x10-bit @ 100ksps; 1D/A, 1x5-bit @ 100ksps, CTMU	2-Comparators, Bandgap - No; 1A/D, 10x10-bit @ 100ksps; 0D/A, 0x0-bit @ ksps
Digital Comm. Peripherals	1-UART, 1-A/E/USART, 1-SPI, 1-I2C, 1-MSSP(SPI/I2C)	1-UART, 1-A/E/USART, 1-SPI, 1-I2C, 1-MSSP(SPI/I2C)	1-UART, 1-A/E/USART, 1-SPI, 1-I2C, 1-MSSP(SPI/I2C)	1-UART, 1-A/E/USART, 1-SPI, 1-I2C, 1-MSSP(SPI/I2C)
Connectivity	1-Full Speed-USB 2.0,None, LIN	1-Full Speed-USB 2.0,None, LIN	1-Full Speed-USB 2.0,None, LIN	1-Full Speed-USB 2.0,None, LIN
Capture/Compare PWM Peripherals	1-ECCP, 10-bitPWM	1-ECCP, 10-bitPWM	1-ECCP, 10-bitPWM	2-CCP, 10-bitPWM
Digital Timers	1x8-bit, 3x16-bit	2x8-bit, 2x16-bit	2x8-bit, 2x16-bit	1x8-bit, 3x16-bit
Application Peripherals	9-mTouch, No, EBI-No	14-mTouch, No, EBI-No	14-mTouch, No, EBI-No	No, EBI-No
Debug/Development Features	ICSP, ICDdebug - With Add On	ICSP, ICDdebug - Integrated	ICSP, ICDdebug - Integrated	ICSP, ICDdebug - Yes
Operating Voltage Temperature Range (°C)	(1.8V-5.5V) (-40 to 125)	(1.8V-5.5V) (-40 to 125)	(1.8V-5.5V) (-40 to 125)	(2V-5.5V) (-40 to 85)
Package Pins	20	28	28	28
Budgetary Price (USD)	$1.53 - $2.41	$1.65 - $2.51	$1.76 - $2.66	$3.44 - $4.51

Figura A.1: Tabella di comparazione delle caratteristiche dei μC Microchip®.

TABLE 28-28: A/D CONVERTER CHARACTERISTICS: PIC18F2455/2550/4455/4550 (INDUSTRIAL) PIC18LF2455/2550/4455/4550 (INDUSTRIAL)

Param No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
A01	NR	Resolution	—	—	10	bit	ΔVREF ≥ 3.0V
A03	EIL	Integral Linearity Error	—	—	<±1	LSb	ΔVREF ≥ 3.0V
A04	EDL	Differential Linearity Error	—	—	<±1	LSb	ΔVREF ≥ 3.0V
A06	EOFF	Offset Error	—	—	<±1.5	LSb	ΔVREF ≥ 3.0V
A07	EGN	Gain Error	—	—	<±1	LSb	ΔVREF ≥ 3.0V
A10	—	Monotonicity	Guaranteed(1)			—	VSS ≤ VAIN ≤ VREF
A20	ΔVREF	Reference Voltage Range (VREFH – VREFL)	1.8 3	— —	— —	V V	VDD < 3.0V VDD ≥ 3.0V
A21	VREFH	Reference Voltage High	VSS	—	VREFH	V	
A22	VREFL	Reference Voltage Low	VSS – 0.3V	—	VDD – 3.0V	V	
A25	VAIN	Analog Input Voltage	VREFL	—	VREFH	V	
A30	ZAIN	Recommended Impedance of Analog Voltage Source	—	—	2.5	kΩ	
A50	IREF	VREF Input Current(2)	— —	— —	5 150	μA μA	During VAIN acquisition. During A/D conversion cycle.

Note 1: The A/D conversion result never decreases with an increase in the input voltage and has no missing codes.

2: VREFH current is from RA3/AN3/VREF+ pin or VDD, whichever is selected as the VREFH source. VREFL current is from RA2/AN2/VREF-/CVREF pin or VSS, whichever is selected as the VREFL source.

Silicon Errata

TABLE 28-8: A/D CONVERTER CHARACTERISTICS:PIC18F2455/2550/4455/4550 (INDUSTRIAL) PIC18LF2455/2550/4455/4550 (INDUSTRIAL)

Param No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
A06A	**EOFF**	**Offset Error**	**—**	**—**	**<±2.0**	**LSb**	**VREF = VREF+ and VREF-**
A06	EOFF	Offset Error	—	—	**<±3.5**	LSb	**VREF = VSS and VDD**

Figura A.2: Caratteristiche dell'ADC integrato nel Microchip® PIC18F2550. (fonte [21] [22])

TABLE 28-29: A/D CONVERSION REQUIREMENTS

Param No.	Symbol	Characteristic		Min	Max	Units	Conditions
130	TAD	A/D Clock Period	PIC18**F**XXXX	0.7	25.0(1)	µs	TOSC based, VREF ≥ 3.0V
			PIC18**LF**XXXX	1.4	25.0(1)	µs	VDD = 2.0V, TOSC based, VREF full range
			PIC18**F**XXXX	TBD	1	µs	A/D RC mode
			PIC18**LF**XXXX	TBD	3	µs	VDD = 2.0V, A/D RC mode
131	TCNV	Conversion Time (not including acquisition time)(2)		11	12	TAD	
132	TACQ	Acquisition Time(3)		1.4 TBD	— —	µs µs	-40°C to +85°C 0°C ≤ to ≤ +85°C
135	TSWC	Switching Time from Convert → Sample		—	**(Note 4)**		
137	TDIS	Discharge Time		0.2	—	µs	

Legend: TBD = To Be Determined

Note 1: The time of the A/D clock period is dependent on the device frequency and the TAD clock divider.

2: ADRES registers may be read on the following TCY cycle.

3: The time for the holding capacitor to acquire the "New" input voltage when the voltage changes full scale after the conversion (VDD to VSS or VSS to VDD). The source impedance (*Rs*) on the input channels is 50Ω.

4: On the following cycle of the device clock.

Figura A.3: Requisiti di conversione dell'ADC integrato nel Microchip® PIC18F2550. (fonte [21])

A.1 Comparazione DAC

Nei grafici seguenti sono riportati:

- in ascissa, i valori `ADC` nell'intervallo $[0, 2^k]$ trasmessi al DAC;
- in ordinata, a sinistra, i valori `DAC` nell'intervallo $[0, 2^k]$ letti dall'ADC;
- in ordinata, a destra, la differenza `DELTA` tra i due precedenti valori;
- l'equazione della retta che interpola i punti `ADC-DAC`.

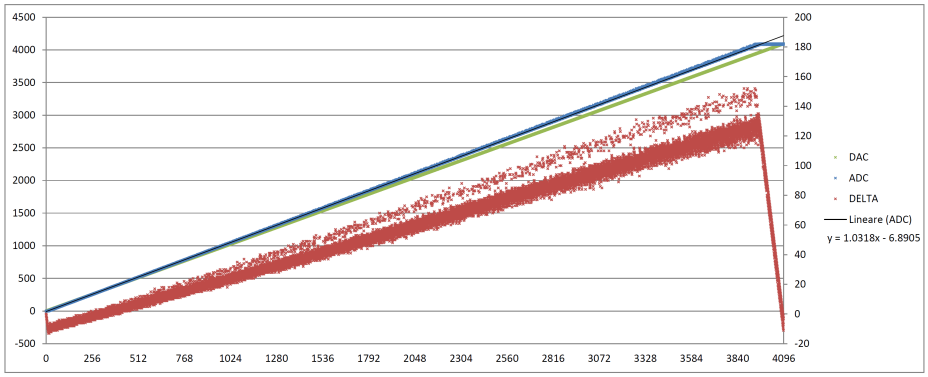


Figura A.4: Analog® AD5620: test di verifica della linearità.

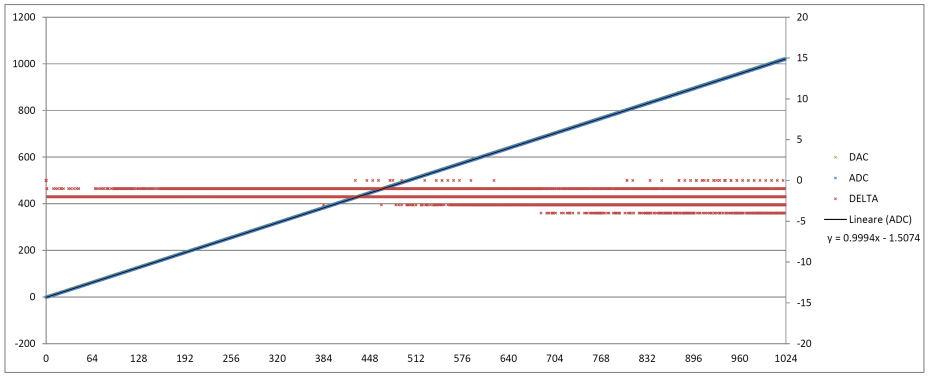


Figura A.5: Texas® DAC101C085: test di verifica della linearità.

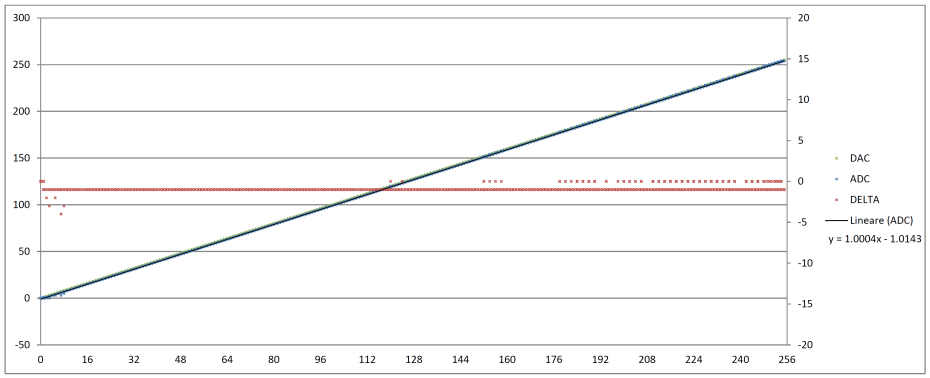


Figura A.6: Maxim® MAX517: test di verifica della linearità.

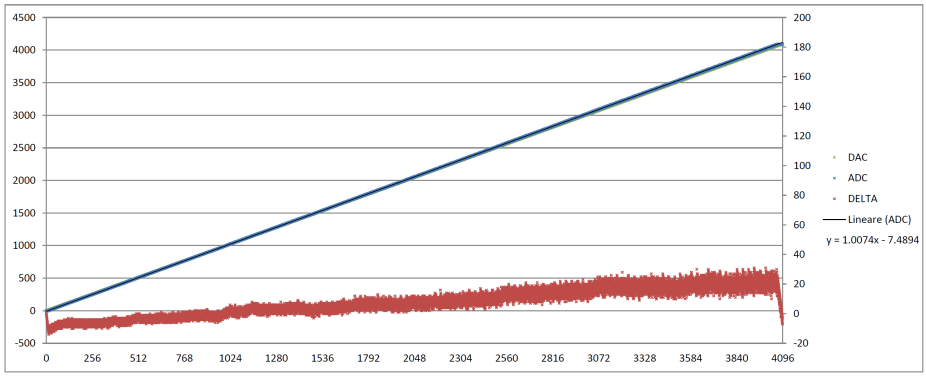


Figura A.7: Microchip® MCP4725: test di verifica della linearità.

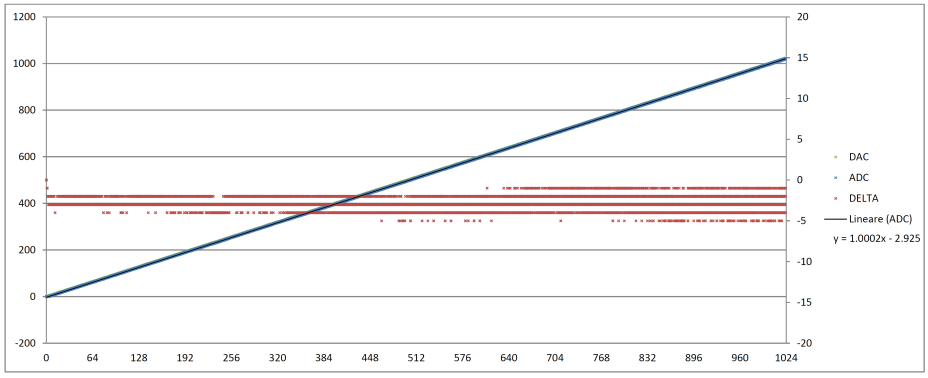


Figura A.8: Microchip® MCP4911: test di verifica della linearità.

A.2 Comparazione S/H

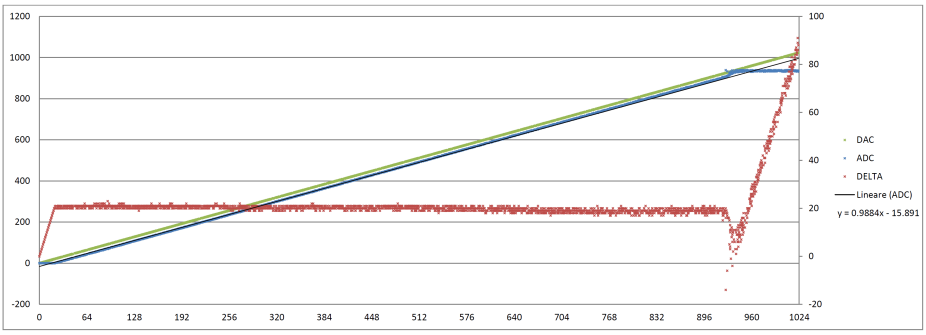


Figura A.9: Texas® LMV601 + LMV602: test di verifica del blocco di ritardo.

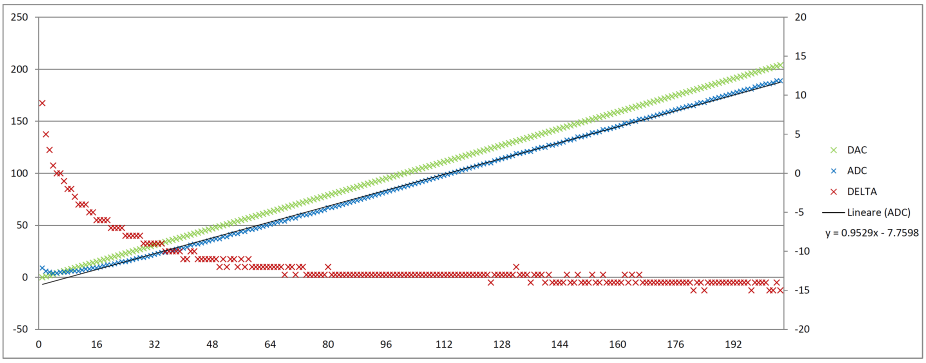


Figura A.10: Maxim® DS1843: test di verifica del blocco di ritardo.

A.3 Schematico

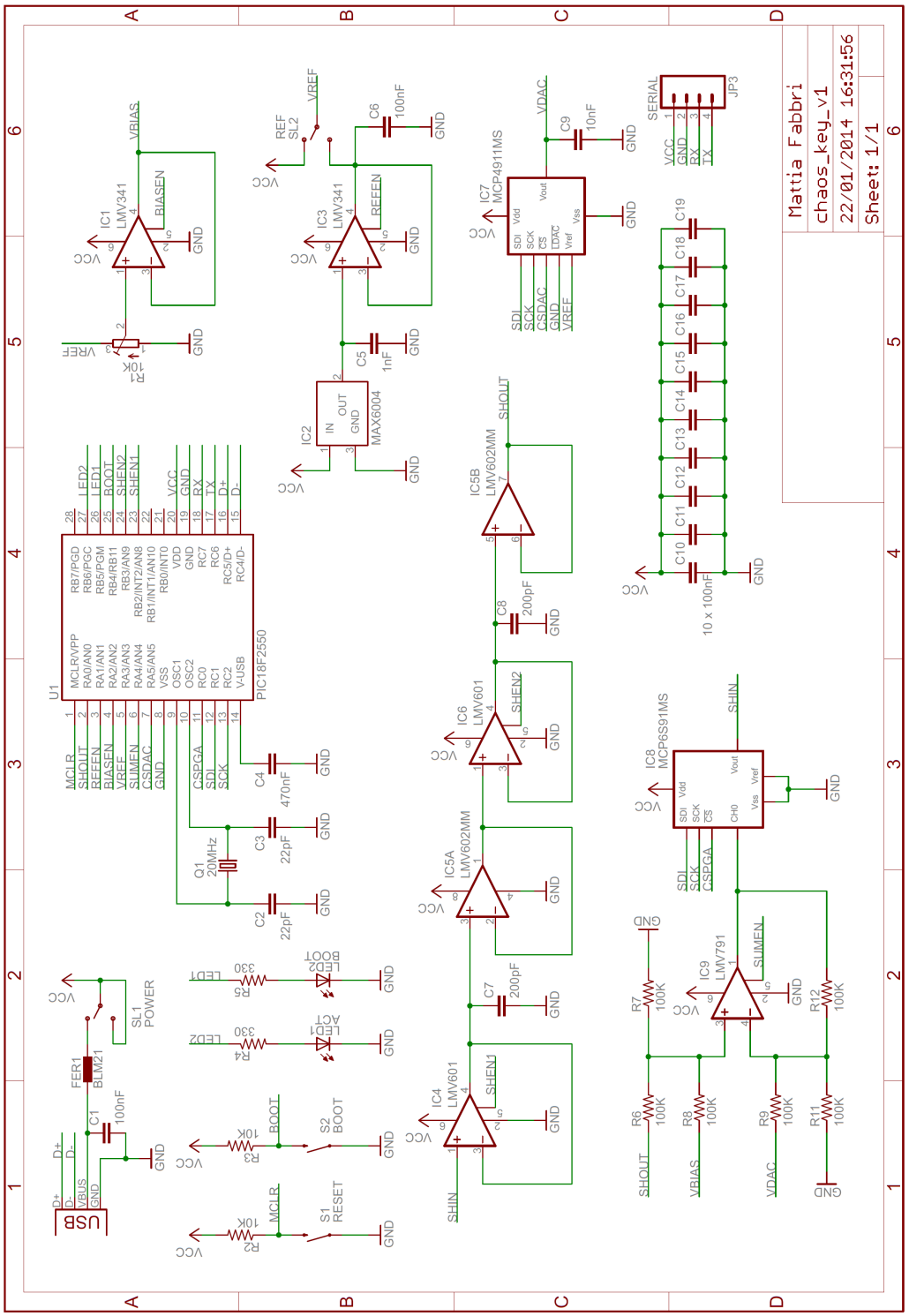


Figura A.11: Schema elettrico del prototipo TRNG basato su dinamica caotica.

A.4 Risultati test statistici NIST - Caso A

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <device_raw_N3K4F1.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE		PROPORTION		STATISTICAL TEST
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	Frequency
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	BlockFrequency
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	CumulativeSums
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	CumulativeSums
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	Runs
24	10	12	12	10	9	8	6	6	3	0.000648		100/100		LongestRun
12	10	8	10	12	10	6	13	9	10	0.924076		99/100		Rank
9	11	14	10	7	8	10	14	15	2	0.137282		98/100		FFT
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
35	18	15	12	3	4	0	4	6	3	0.000000	*	87/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
71	15	7	5	0	1	1	0	0	0	0.000000	*	61/100	*	NonOverlappingTemplate
30	10	10	11	9	3	9	8	4	6	0.000000	*	94/100	*	NonOverlappingTemplate
85	7	3	1	1	0	1	0	1	1	0.000000	*	56/100	*	NonOverlappingTemplate
83	4	4	3	1	3	0	0	2	0	0.000000	*	46/100	*	NonOverlappingTemplate
69	10	8	3	4	4	0	1	0	1	0.000000	*	67/100	*	NonOverlappingTemplate
58	14	9	5	5	4	2	2	0	1	0.000000	*	74/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
98	2	0	0	0	0	0	0	0	0	0.000000	*	9/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
85	10	2	1	1	0	0	1	0	0	0.000000	*	36/100	*	NonOverlappingTemplate
13	14	7	12	9	16	8	8	6	7	0.289667		99/100		NonOverlappingTemplate
10	15	9	9	8	8	5	10	10	16	0.383827		100/100		NonOverlappingTemplate
21	15	14	13	9	6	6	7	3	6	0.001030		97/100		NonOverlappingTemplate
17	11	10	7	7	4	14	13	10	7	0.129620		95/100	*	NonOverlappingTemplate
8	14	13	8	8	8	13	12	10	6	0.637119		99/100		NonOverlappingTemplate
8	15	8	15	14	13	9	5	6	7	0.145326		98/100		NonOverlappingTemplate
12	12	13	12	7	12	10	9	7	6	0.739918		100/100		NonOverlappingTemplate
8	10	14	11	9	11	7	13	7	10	0.834308		99/100		NonOverlappingTemplate
11	9	10	7	15	6	4	18	11	9	0.080519		100/100		NonOverlappingTemplate
43	14	10	9	3	7	5	4	5	0	0.000000	*	90/100	*	NonOverlappingTemplate
62	10	11	9	3	2	2	1	0	0	0.000000	*	80/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate

100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
21	9	20	11	11	9	10	3	3	3	0.000024	*	96/100	NonOverlappingTemplate
76	13	3	4	3	1	0	0	0	0	0.000000	*	53/100	* NonOverlappingTemplate
83	7	4	3	1	1	1	0	0	0	0.000000	*	56/100	* NonOverlappingTemplate
70	12	6	5	2	3	1	1	0	0	0.000000	*	69/100	* NonOverlappingTemplate
11	13	12	6	9	14	4	10	10	11	0.494392		99/100	NonOverlappingTemplate
42	11	12	10	9	3	6	3	2	2	0.000000	*	89/100	* NonOverlappingTemplate
30	13	16	8	6	4	9	7	4	3	0.000000	*	87/100	* NonOverlappingTemplate
11	10	11	9	11	11	10	11	9	7	0.996335		99/100	NonOverlappingTemplate
82	7	2	1	3	1	2	1	1	0	0.000000	*	50/100	* NonOverlappingTemplate
16	11	8	9	12	12	7	14	8	3	0.171867		99/100	NonOverlappingTemplate
16	14	12	9	14	8	7	9	2	9	0.085587		93/100	* NonOverlappingTemplate
64	8	12	6	2	3	2	2	0	1	0.000000	*	72/100	* NonOverlappingTemplate
13	10	11	9	7	11	6	10	8	15	0.678686		99/100	NonOverlappingTemplate
32	21	9	9	6	4	8	7	3	1	0.000000	*	93/100	* NonOverlappingTemplate
32	11	17	11	9	6	5	2	4	3	0.000000	*	93/100	* NonOverlappingTemplate
16	11	15	10	9	10	5	7	8	9	0.334538		99/100	NonOverlappingTemplate
16	12	14	11	4	15	3	11	10	4	0.015598		99/100	NonOverlappingTemplate
14	12	9	9	12	13	6	8	14	3	0.213309		100/100	NonOverlappingTemplate
69	15	3	6	2	1	2	1	0	1	0.000000	*	70/100	* NonOverlappingTemplate
94	3	1	1	0	1	0	0	0	0	0.000000	*	30/100	* NonOverlappingTemplate
38	17	8	9	8	0	12	2	3	3	0.000000	*	95/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
96	3	1	0	0	0	0	0	0	0	0.000000	*	24/100	* NonOverlappingTemplate
13	12	12	7	13	9	11	10	7	6	0.719747		97/100	NonOverlappingTemplate
19	13	15	5	10	7	6	8	11	6	0.028817		97/100	NonOverlappingTemplate
13	12	12	13	11	14	5	9	4	7	0.249284		96/100	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
69	14	6	6	1	1	1	2	0	0	0.000000	*	78/100	* NonOverlappingTemplate
42	19	13	3	7	8	1	3	2	2	0.000000	*	86/100	* NonOverlappingTemplate
95	2	2	1	0	0	0	0	0	0	0.000000	*	20/100	* NonOverlappingTemplate
84	6	5	2	1	1	0	1	0	0	0.000000	*	48/100	* NonOverlappingTemplate
92	5	2	1	0	0	0	0	0	0	0.000000	*	20/100	* NonOverlappingTemplate
68	7	9	8	3	0	2	1	0	2	0.000000	*	68/100	* NonOverlappingTemplate
99	1	0	0	0	0	0	0	0	0	0.000000	*	5/100	* NonOverlappingTemplate
95	2	3	0	0	0	0	0	0	0	0.000000	*	26/100	* NonOverlappingTemplate
81	7	7	1	1	1	1	1	0	0	0.000000	*	48/100	* NonOverlappingTemplate
91	2	2	2	1	0	1	1	0	0	0.000000	*	28/100	* NonOverlappingTemplate
99	0	0	1	0	0	0	0	0	0	0.000000	*	5/100	* NonOverlappingTemplate
91	5	0	1	2	1	0	0	0	0	0.000000	*	35/100	* NonOverlappingTemplate
99	0	1	0	0	0	0	0	0	0	0.000000	*	2/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
97	3	0	0	0	0	0	0	0	0	0.000000	*	13/100	* NonOverlappingTemplate
96	1	1	1	1	0	0	0	0	0	0.000000	*	24/100	* NonOverlappingTemplate
70	14	3	4	2	4	1	1	0	1	0.000000	*	57/100	* NonOverlappingTemplate
41	14	11	8	5	8	4	5	3	1	0.000000	*	92/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate

```
 95   4   1   0   0   0   0   0   0   0 0.000000 *   20/100  *  NonOverlappingTemplate
 47  23   9   8   5   5   1   1   1   0 0.000000 *   87/100  *  NonOverlappingTemplate
 12  10  11   8   9  11   3   9  13  14 0.474986     97/100     NonOverlappingTemplate
 90   5   3   2   0   0   0   0   0   0 0.000000 *   33/100  *  NonOverlappingTemplate
 10  10  10  10   9   5  11  14  13   8 0.779188     99/100     NonOverlappingTemplate
 29  17  17   4   6   7   9   2   6   3 0.000000 *   92/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
 11  10  10  13  10   8   8  14   3  13 0.419021     98/100     NonOverlappingTemplate
 72   7   8   6   4   2   0   1   0   0 0.000000 *   64/100  *  NonOverlappingTemplate
 25  12   7  13  21   5   0   6   6   5 0.000000 *   94/100  *  NonOverlappingTemplate
 76   8   3   5   2   1   2   1   1   1 0.000000 *   62/100  *  NonOverlappingTemplate
 52  19   8   6   3   3   2   2   2   3 0.000000 *   91/100  *  NonOverlappingTemplate
 47  16   8  12   5   3   2   4   1   2 0.000000 *   81/100  *  NonOverlappingTemplate
 99   0   1   0   0   0   0   0   0   0 0.000000 *    5/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
 82   9   4   0   1   2   1   0   1   0 0.000000 *   53/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
 49  15  13   6   4   8   3   1   1   0 0.000000 *   87/100  *  NonOverlappingTemplate
 45  13  10   7   8   6   2   2   4   3 0.000000 *   82/100  *  NonOverlappingTemplate
  9  10  13  13  14   7  11   7   5  11 0.534146     99/100     NonOverlappingTemplate
  9   8  15  14  10   7  11  14   7   5 0.304126     98/100     NonOverlappingTemplate
 52  15  11   6   3   1   7   3   1   1 0.000000 *   84/100  *  NonOverlappingTemplate
 65  11  11   3   3   4   0   0   2   1 0.000000 *   64/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
 77  13   6   1   0   2   0   1   0   0 0.000000 *   56/100  *  NonOverlappingTemplate
 95   4   1   0   0   0   0   0   0   0 0.000000 *   34/100  *  NonOverlappingTemplate
 18  18  11   9  12   9   3   6   7   7 0.009535     99/100     NonOverlappingTemplate
 59  15   6   5   3   6   4   2   0   0 0.000000 *   77/100  *  NonOverlappingTemplate
 88   4   1   2   3   0   2   0   0   0 0.000000 *   44/100  *  NonOverlappingTemplate
 21  13  17   9   6   5   9   6   7   7 0.002374     96/100     NonOverlappingTemplate
 19  20  11  10  12   8   5   7   7   1 0.000253     97/100     NonOverlappingTemplate
 33  12  13   7   9   3   6   3   9   5 0.000000 *   88/100  *  NonOverlappingTemplate
 10   8  10   9   8  11   8   7  16  13 0.657933     98/100     NonOverlappingTemplate
 88   4   1   0   4   2   1   0   0   0 0.000000 *   46/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  NonOverlappingTemplate
 88   6   3   0   1   0   2   0   0   0 0.000000 *   49/100  *  NonOverlappingTemplate
 92   4   2   1   0   0   1   0   0   0 0.000000 *   46/100  *  NonOverlappingTemplate
 12  19  14  14   6   8  10   6   4   7 0.019188    100/100     NonOverlappingTemplate
 30  14   7  10   5   9  10   5   6   4 0.000000 *   94/100  *  NonOverlappingTemplate
 17  22  11   9   9   9   6   6   4   7 0.001201     95/100  *  NonOverlappingTemplate
 97   1   0   0   1   1   0   0   0   0 0.000000 *   20/100  *  NonOverlappingTemplate
 22  16  10   6   9   7  13   7   7   3 0.000883     94/100  *  NonOverlappingTemplate
 39  12  10   9   4   6  12   3   4   1 0.000000 *   88/100  *  NonOverlappingTemplate
 61  16   7   7   3   4   2   0   0   0 0.000000 *   73/100  *  NonOverlappingTemplate
 57  14  12   4   6   4   0   1   2   0 0.000000 *   82/100  *  NonOverlappingTemplate
 14   9   1  16  14  11  14  11   3   7 0.007160     98/100     NonOverlappingTemplate
 12  16  16  11   9   7  11   3   4  11 0.042808     97/100     NonOverlappingTemplate
 66  12   6   6   3   3   2   0   0   2 0.000000 *   76/100  *  NonOverlappingTemplate
```

100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
10	8	6	14	8	13	12	8	14	7	0.514124		100/100	NonOverlappingTemplate
16	8	8	12	17	7	8	12	3	9	0.058984		99/100	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
55	16	8	6	5	2	2	2	2	2	0.000000	*	80/100	* NonOverlappingTemplate
3	10	15	11	15	7	8	10	12	9	0.224821		100/100	NonOverlappingTemplate
65	16	6	4	3	1	3	1	1	0	0.000000	*	72/100	* NonOverlappingTemplate
95	4	1	0	0	0	0	0	0	0	0.000000	*	31/100	* NonOverlappingTemplate
98	2	0	0	0	0	0	0	0	0	0.000000	*	8/100	* NonOverlappingTemplate
84	6	2	1	4	1	2	0	0	0	0.000000	*	39/100	* NonOverlappingTemplate
14	10	6	22	6	9	13	4	9	7	0.003201		99/100	NonOverlappingTemplate
92	5	2	1	0	0	0	0	0	0	0.000000	*	24/100	* NonOverlappingTemplate
94	4	0	0	0	1	0	1	0	0	0.000000	*	35/100	* NonOverlappingTemplate
25	21	6	11	7	7	10	5	7	1	0.000000	*	95/100	* NonOverlappingTemplate
99	1	0	0	0	0	0	0	0	0	0.000000	*	3/100	* NonOverlappingTemplate
99	0	1	0	0	0	0	0	0	0	0.000000	*	2/100	* NonOverlappingTemplate
89	3	4	3	1	0	0	0	0	0	0.000000	*	36/100	* OverlappingTemplate
82	9	2	3	1	2	0	0	1	0	0.000000	*	57/100	* Universal
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* ApproximateEntropy
0	0	0	0	0	0	1	0	0	0	----		1/1	RandomExcursions
0	0	0	0	0	0	0	1	0	0	----		1/1	RandomExcursions
0	0	0	0	0	1	0	0	0	0	----		1/1	RandomExcursions
0	0	0	0	0	0	1	0	0	0	----		1/1	RandomExcursions
0	0	0	0	0	1	0	0	0	0	----		1/1	RandomExcursions
0	1	0	0	0	0	0	0	0	0	----		1/1	RandomExcursions
0	0	0	1	0	0	0	0	0	0	----		1/1	RandomExcursions
0	0	0	1	0	0	0	0	0	0	----		1/1	RandomExcursions
1	0	0	0	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
1	0	0	0	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
1	0	0	0	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	1	0	0	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	0	0	1	0	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	0	0	1	0	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	1	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	0	0	0	1	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	0	0	1	0	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	0	1	0	0	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	0	0	0	1	----		1/1	RandomExcursionsVariant
0	0	0	0	0	0	1	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	1	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	0	1	0	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	1	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	1	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	1	0	0	0	0	0	0	----		1/1	RandomExcursionsVariant
0	0	0	0	1	0	0	0	0	0	----		1/1	RandomExcursionsVariant
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* Serial
85	4	4	4	0	2	1	0	0	0	0.000000	*	55/100	* Serial
7	11	8	3	11	9	15	10	12	14	0.275709		100/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 0 for a sample size = 1 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.1: Risultati del test NIST STS 2.1 su una sequenza di $100 \times 10^6\,\mathrm{bit}$ generati dal TRNG su mappa ADCM a 4 rami, come mostrato in Fig. 5.8. I 2 bit prodotti ad ogni iterazione della mappa vengono memorizzati e poi inviati senza effettuare alcun post-processing. Lo schema di codifica impiegato è 00-01-10-11-00. La sequenza analizzata è prelevata in ingresso all'entropy distiller di Linux™.

A.5 Risultati test statistici NIST - Caso B

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <device_raw_N3K4F1_pool.txt>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
8	10	6	12	12	10	11	10	9	12	0.946308	100/100	Frequency
10	8	8	11	9	11	18	8	8	9	0.494392	97/100	BlockFrequency
6	9	7	9	11	16	15	9	10	8	0.401199	100/100	CumulativeSums
7	15	8	5	11	8	12	15	11	8	0.334538	100/100	CumulativeSums
14	9	10	4	13	14	15	9	7	5	0.129620	99/100	Runs
7	11	9	12	12	10	12	14	5	8	0.657933	99/100	LongestRun
11	8	9	8	10	12	12	10	11	9	0.991468	99/100	Rank
10	17	8	7	11	8	10	10	5	14	0.289667	99/100	FFT
8	9	14	13	9	8	6	10	15	8	0.534146	98/100	NonOverlappingTemplate
7	8	12	12	10	14	6	18	7	6	0.115387	100/100	NonOverlappingTemplate
8	11	7	9	7	14	12	13	9	10	0.798139	100/100	NonOverlappingTemplate
11	13	8	11	9	12	13	4	8	11	0.637119	100/100	NonOverlappingTemplate
10	12	10	6	9	10	13	13	10	7	0.851383	100/100	NonOverlappingTemplate
6	11	9	16	10	6	10	13	7	12	0.419021	99/100	NonOverlappingTemplate
13	3	7	17	10	9	6	12	8	15	0.055361	100/100	NonOverlappingTemplate
17	12	7	8	5	11	8	8	10	14	0.236810	100/100	NonOverlappingTemplate
11	9	9	11	12	7	7	14	13	7	0.739918	97/100	NonOverlappingTemplate
7	11	12	11	11	10	5	13	11	9	0.816537	98/100	NonOverlappingTemplate
14	11	7	10	10	9	11	6	7	15	0.554420	98/100	NonOverlappingTemplate
10	5	7	12	16	9	5	6	14	16	0.051942	99/100	NonOverlappingTemplate
14	16	7	12	8	14	4	13	5	7	0.058984	99/100	NonOverlappingTemplate
5	9	14	12	12	12	9	10	8	9	0.739918	100/100	NonOverlappingTemplate
10	6	15	11	10	7	10	9	11	11	0.798139	96/100	NonOverlappingTemplate
13	10	6	15	12	6	10	10	11	7	0.534146	99/100	NonOverlappingTemplate
8	11	12	11	5	14	7	15	7	10	0.401199	99/100	NonOverlappingTemplate
15	10	11	11	5	7	5	15	13	8	0.191687	99/100	NonOverlappingTemplate
10	12	6	9	5	12	10	14	13	9	0.574903	98/100	NonOverlappingTemplate
9	12	10	15	9	8	13	7	9	8	0.759756	99/100	NonOverlappingTemplate
11	8	11	6	11	3	14	16	9	11	0.181557	100/100	NonOverlappingTemplate
5	12	14	8	13	9	8	15	6	10	0.319084	99/100	NonOverlappingTemplate
11	10	7	6	9	12	9	10	13	13	0.834308	100/100	NonOverlappingTemplate
13	9	8	9	11	9	10	8	16	7	0.678686	98/100	NonOverlappingTemplate
13	12	8	10	5	11	8	10	16	7	0.419021	99/100	NonOverlappingTemplate
13	11	12	10	7	12	11	2	9	13	0.334538	99/100	NonOverlappingTemplate
10	8	13	11	12	10	8	7	9	12	0.935716	98/100	NonOverlappingTemplate
8	13	9	13	9	13	6	11	7	11	0.739918	99/100	NonOverlappingTemplate
9	11	4	7	15	12	10	6	10	16	0.171867	100/100	NonOverlappingTemplate
7	11	14	10	10	11	12	9	13	3	0.437274	100/100	NonOverlappingTemplate
12	9	11	12	9	12	8	13	9	5	0.798139	100/100	NonOverlappingTemplate
14	13	14	7	11	13	6	5	10	7	0.275709	98/100	NonOverlappingTemplate

7	12	7	10	11	12	17	8	7	9	0.437274	98/100	NonOverlappingTemplate
7	11	7	9	10	8	8	18	9	13	0.334538	99/100	NonOverlappingTemplate
12	9	3	10	12	9	15	13	7	10	0.334538	100/100	NonOverlappingTemplate
12	8	6	13	8	7	11	10	10	15	0.616305	100/100	NonOverlappingTemplate
16	13	12	9	7	6	7	5	10	15	0.145326	98/100	NonOverlappingTemplate
10	9	14	12	10	11	9	8	5	12	0.779188	100/100	NonOverlappingTemplate
5	10	10	11	12	10	7	17	12	6	0.289667	100/100	NonOverlappingTemplate
7	6	8	9	11	10	14	10	13	12	0.739918	98/100	NonOverlappingTemplate
12	10	13	12	13	14	7	5	5	9	0.334538	98/100	NonOverlappingTemplate
12	10	7	8	9	10	9	13	8	14	0.851383	99/100	NonOverlappingTemplate
10	4	10	14	10	11	10	14	10	7	0.554420	98/100	NonOverlappingTemplate
16	8	14	7	11	7	7	4	14	12	0.122325	98/100	NonOverlappingTemplate
12	12	9	6	3	8	16	8	15	11	0.108791	99/100	NonOverlappingTemplate
10	8	13	10	13	11	9	5	9	12	0.798139	97/100	NonOverlappingTemplate
16	9	8	11	12	13	8	14	4	5	0.137282	97/100	NonOverlappingTemplate
16	10	12	13	4	5	10	14	9	7	0.137282	98/100	NonOverlappingTemplate
20	9	11	7	11	11	7	6	14	4	0.025193	100/100	NonOverlappingTemplate
10	8	9	16	6	16	9	9	11	6	0.262249	99/100	NonOverlappingTemplate
9	11	3	13	11	9	10	9	11	14	0.534146	99/100	NonOverlappingTemplate
9	10	7	14	11	5	13	10	8	13	0.595549	100/100	NonOverlappingTemplate
16	5	16	9	7	10	5	13	11	8	0.102526	100/100	NonOverlappingTemplate
10	12	12	12	15	6	7	10	5	11	0.455937	99/100	NonOverlappingTemplate
8	9	5	7	17	11	18	8	12	5	0.028817	100/100	NonOverlappingTemplate
10	6	11	7	9	9	16	12	9	11	0.637119	97/100	NonOverlappingTemplate
10	7	7	13	10	14	9	9	16	5	0.304126	100/100	NonOverlappingTemplate
6	14	13	11	12	8	8	12	11	5	0.494392	100/100	NonOverlappingTemplate
13	8	10	13	12	7	6	7	9	15	0.474986	97/100	NonOverlappingTemplate
16	12	11	4	9	13	8	9	5	13	0.181557	99/100	NonOverlappingTemplate
11	13	12	13	4	13	13	8	9	4	0.224821	100/100	NonOverlappingTemplate
8	8	9	10	16	11	13	8	8	9	0.699313	99/100	NonOverlappingTemplate
9	11	11	12	15	12	4	11	9	6	0.437274	99/100	NonOverlappingTemplate
6	19	8	6	9	12	6	10	14	10	0.080519	100/100	NonOverlappingTemplate
6	16	9	16	13	5	7	7	8	13	0.080519	97/100	NonOverlappingTemplate
14	6	9	12	8	11	13	6	13	8	0.534146	100/100	NonOverlappingTemplate
11	11	11	6	12	9	8	13	5	14	0.554420	100/100	NonOverlappingTemplate
16	8	6	11	9	9	8	10	13	10	0.616305	100/100	NonOverlappingTemplate
10	6	15	14	6	8	9	8	14	10	0.366918	97/100	NonOverlappingTemplate
8	18	5	14	7	12	8	7	15	6	0.040108	98/100	NonOverlappingTemplate
7	6	6	10	11	14	11	12	13	10	0.616305	98/100	NonOverlappingTemplate
13	17	11	10	7	11	10	5	6	10	0.275709	99/100	NonOverlappingTemplate
12	12	10	7	14	12	8	8	7	10	0.798139	98/100	NonOverlappingTemplate
9	9	11	10	10	13	11	7	9	11	0.983453	99/100	NonOverlappingTemplate
8	9	14	13	9	8	6	10	15	8	0.534146	98/100	NonOverlappingTemplate
16	7	6	9	7	11	10	14	8	12	0.383827	99/100	NonOverlappingTemplate
9	9	5	12	8	12	12	10	13	10	0.816537	99/100	NonOverlappingTemplate
12	5	9	7	11	13	16	10	10	7	0.401199	100/100	NonOverlappingTemplate
11	8	12	9	14	11	10	7	14	4	0.455937	99/100	NonOverlappingTemplate
9	11	9	11	9	14	16	11	3	7	0.236810	100/100	NonOverlappingTemplate
10	10	3	11	11	7	18	9	11	10	0.181557	100/100	NonOverlappingTemplate
5	4	12	12	9	15	9	13	8	13	0.224821	100/100	NonOverlappingTemplate

8	10	6	15	10	9	15	8	10	9	0.574903	99/100	NonOverlappingTemplate
15	10	12	14	7	9	6	8	10	9	0.574903	99/100	NonOverlappingTemplate
18	13	7	10	9	8	11	7	9	8	0.334538	96/100	NonOverlappingTemplate
8	15	13	10	13	9	9	13	5	5	0.289667	100/100	NonOverlappingTemplate
10	11	7	9	10	8	11	8	11	15	0.867692	99/100	NonOverlappingTemplate
7	10	11	7	11	13	13	9	6	13	0.699313	99/100	NonOverlappingTemplate
13	3	13	11	6	11	9	10	12	12	0.401199	100/100	NonOverlappingTemplate
9	14	8	17	11	7	6	10	9	9	0.366918	99/100	NonOverlappingTemplate
15	6	10	8	15	12	5	11	9	9	0.334538	99/100	NonOverlappingTemplate
8	6	15	12	11	15	10	8	8	7	0.419021	100/100	NonOverlappingTemplate
10	7	12	11	10	6	9	16	13	6	0.419021	97/100	NonOverlappingTemplate
11	9	7	14	7	11	3	12	11	15	0.236810	99/100	NonOverlappingTemplate
14	10	9	9	9	12	12	8	7	10	0.911413	98/100	NonOverlappingTemplate
6	3	13	15	8	11	10	17	8	9	0.071177	100/100	NonOverlappingTemplate
8	12	8	10	8	10	8	16	15	5	0.304126	100/100	NonOverlappingTemplate
8	12	11	13	11	13	8	6	11	7	0.759756	99/100	NonOverlappingTemplate
11	7	13	9	12	8	15	11	5	9	0.534146	97/100	NonOverlappingTemplate
12	3	16	12	7	14	11	7	13	5	0.062821	99/100	NonOverlappingTemplate
8	14	9	12	9	11	10	14	5	8	0.616305	99/100	NonOverlappingTemplate
15	9	4	10	7	13	10	9	12	11	0.474986	99/100	NonOverlappingTemplate
14	4	11	14	5	12	7	10	13	10	0.236810	98/100	NonOverlappingTemplate
9	10	10	7	7	16	5	24	5	7	0.000296	100/100	NonOverlappingTemplate
15	10	8	11	7	12	7	7	12	11	0.678686	99/100	NonOverlappingTemplate
4	10	14	10	3	8	14	5	22	10	0.000648	98/100	NonOverlappingTemplate
8	11	6	10	9	9	16	13	9	9	0.637119	100/100	NonOverlappingTemplate
11	9	5	11	7	10	16	10	12	9	0.554420	100/100	NonOverlappingTemplate
2	11	10	4	13	7	15	18	14	6	0.004301	100/100	NonOverlappingTemplate
8	9	11	12	9	11	12	12	9	7	0.964295	99/100	NonOverlappingTemplate
9	11	8	9	15	9	12	12	7	8	0.798139	99/100	NonOverlappingTemplate
10	6	10	14	8	17	8	12	9	6	0.275709	99/100	NonOverlappingTemplate
14	11	8	10	9	9	8	10	11	10	0.971699	98/100	NonOverlappingTemplate
11	14	5	9	7	13	13	5	8	15	0.191687	99/100	NonOverlappingTemplate
6	9	9	11	9	5	21	13	8	9	0.035174	99/100	NonOverlappingTemplate
6	11	9	9	10	10	15	14	8	8	0.657933	100/100	NonOverlappingTemplate
10	11	17	9	12	9	10	3	10	9	0.304126	96/100	NonOverlappingTemplate
11	15	12	7	8	11	8	8	11	9	0.798139	98/100	NonOverlappingTemplate
7	13	9	9	11	10	12	7	11	11	0.935716	100/100	NonOverlappingTemplate
6	14	7	11	10	10	11	11	9	11	0.867692	98/100	NonOverlappingTemplate
9	7	15	9	11	11	10	7	13	8	0.739918	100/100	NonOverlappingTemplate
12	6	12	16	6	7	9	13	8	11	0.350485	97/100	NonOverlappingTemplate
10	7	6	16	8	12	11	5	11	14	0.262249	99/100	NonOverlappingTemplate
7	6	7	14	11	11	15	11	7	11	0.455937	100/100	NonOverlappingTemplate
8	8	15	11	7	13	10	7	11	10	0.719747	98/100	NonOverlappingTemplate
11	5	10	14	7	13	12	9	11	8	0.637119	99/100	NonOverlappingTemplate
8	10	4	10	17	10	10	12	8	11	0.366918	99/100	NonOverlappingTemplate
9	10	11	12	11	15	1	11	9	11	0.236810	100/100	NonOverlappingTemplate
9	21	5	9	10	11	11	6	9	9	0.051942	99/100	NonOverlappingTemplate
13	10	9	11	12	7	5	11	14	8	0.637119	99/100	NonOverlappingTemplate
10	15	7	10	13	13	9	9	6	8	0.595549	99/100	NonOverlappingTemplate
7	16	10	10	9	7	8	9	9	15	0.474986	99/100	NonOverlappingTemplate

11	11	9	11	8	14	12	11	10	3	0.554420	98/100	NonOverlappingTemplate
13	11	12	11	8	4	7	10	13	11	0.595549	96/100	NonOverlappingTemplate
8	11	4	10	12	11	14	12	13	5	0.350485	100/100	NonOverlappingTemplate
8	6	20	10	9	8	11	9	6	13	0.085587	100/100	NonOverlappingTemplate
11	9	12	9	6	11	9	13	12	8	0.897763	99/100	NonOverlappingTemplate
9	10	11	10	8	8	9	16	12	7	0.739918	100/100	NonOverlappingTemplate
11	9	5	10	13	9	13	9	11	10	0.851383	100/100	NonOverlappingTemplate
7	11	13	14	9	8	9	7	12	10	0.798139	100/100	NonOverlappingTemplate
13	13	13	8	7	9	8	9	8	12	0.798139	97/100	NonOverlappingTemplate
8	9	13	10	9	11	8	9	11	12	0.978072	100/100	NonOverlappingTemplate
13	9	10	12	6	8	12	10	8	12	0.867692	98/100	NonOverlappingTemplate
13	13	5	9	12	6	13	10	10	9	0.595549	100/100	NonOverlappingTemplate
8	10	14	8	11	9	8	11	13	8	0.883171	98/100	NonOverlappingTemplate
10	9	17	9	8	10	11	9	8	9	0.719747	99/100	NonOverlappingTemplate
11	7	10	10	10	9	13	11	12	7	0.946308	98/100	NonOverlappingTemplate
9	9	11	10	10	13	11	7	9	11	0.983453	99/100	NonOverlappingTemplate
11	7	7	10	12	13	12	9	11	8	0.897763	99/100	OverlappingTemplate
13	11	9	14	8	12	9	10	3	11	0.474986	98/100	Universal
13	11	17	4	8	11	5	13	11	7	0.108791	99/100	ApproximateEntropy
5	11	4	4	8	7	4	5	8	4	0.468595	59/60	RandomExcursions
2	6	10	9	5	5	8	3	6	6	0.407091	60/60	RandomExcursions
11	4	9	7	9	3	5	6	3	3	0.178278	58/60	RandomExcursions
8	8	4	10	2	7	4	6	2	9	0.195163	60/60	RandomExcursions
7	4	8	5	5	8	6	4	6	7	0.949602	59/60	RandomExcursions
8	11	8	7	4	6	3	3	5	5	0.378138	57/60	RandomExcursions
7	8	6	9	8	4	3	5	2	8	0.468595	59/60	RandomExcursions
10	6	4	5	3	8	6	9	4	5	0.534146	59/60	RandomExcursions
9	5	3	4	7	5	6	11	7	3	0.350485	59/60	RandomExcursionsVariant
8	5	5	5	4	8	5	11	7	2	0.378138	58/60	RandomExcursionsVariant
8	4	7	5	6	7	7	7	7	2	0.834308	57/60	RandomExcursionsVariant
9	5	8	4	6	7	4	6	9	2	0.534146	58/60	RandomExcursionsVariant
9	8	2	8	3	6	7	9	3	5	0.324180	59/60	RandomExcursionsVariant
8	7	6	6	4	5	6	5	6	7	0.991468	58/60	RandomExcursionsVariant
9	5	5	7	7	8	4	4	8	3	0.706149	58/60	RandomExcursionsVariant
8	8	7	2	7	5	8	6	5	4	0.739918	59/60	RandomExcursionsVariant
6	10	7	4	5	5	3	9	6	5	0.637119	60/60	RandomExcursionsVariant
6	4	6	6	7	5	6	6	11	3	0.671779	58/60	RandomExcursionsVariant
4	9	6	8	3	7	7	4	7	5	0.772760	60/60	RandomExcursionsVariant
6	10	3	1	7	7	9	6	5	6	0.324180	60/60	RandomExcursionsVariant
8	7	4	2	3	11	8	5	7	5	0.275709	60/60	RandomExcursionsVariant
9	5	5	5	6	7	6	5	10	2	0.568055	60/60	RandomExcursionsVariant
8	7	7	6	4	4	7	7	6	4	0.949602	60/60	RandomExcursionsVariant
8	10	2	6	6	6	4	8	6	4	0.534146	60/60	RandomExcursionsVariant
12	6	4	3	8	6	7	4	7	3	0.253551	59/60	RandomExcursionsVariant
14	4	3	4	5	7	6	6	6	5	0.122325	59/60	RandomExcursionsVariant
17	9	8	7	9	8	10	10	12	10	0.616305	97/100	Serial
12	5	13	10	11	13	11	8	9	8	0.759756	98/100	Serial
12	9	6	7	11	5	18	10	15	7	0.080519	99/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 57 for a sample size = 60 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.2: Risultati del test NIST STS 2.1 su una sequenza di 100×10^6 bit prelevati in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso A.

A.6 Risultati test statistici NIST - Caso C

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <device_raw_N3K4F1_alt_code.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  Frequency
 60  15   9   8   5   1   0   0   2   0 0.000000 *   81/100  *  BlockFrequency
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  CumulativeSums
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  CumulativeSums
100   0   0   0   0   0   0   0   0   0 0.000000 *    0/100  *  Runs
  9  13  11  15  13  11   6   9   6   7 0.455937     99/100     LongestRun
 10   9   8  10   9   9  11   9  10  15 0.946308    100/100     Rank
 16  17  10  10  12   6  10   3   6  10 0.048716     99/100     FFT
 68  16   6   5   1   1   3   0   0   0 0.000000 *   59/100  *  NonOverlappingTemplate
 57  20   6   8   2   1   3   1   1   1 0.000000 *   76/100  *  NonOverlappingTemplate
 37  11  10  14   5   3   8   6   4   2 0.000000 *   92/100  *  NonOverlappingTemplate
 41  20  10   7   6   5   6   4   1   0 0.000000 *   93/100  *  NonOverlappingTemplate
 29  14  13   9   9   5   9   2   5   5 0.000000 *   95/100  *  NonOverlappingTemplate
 25  11  14  13  10   4  10   5   3   5 0.000014 *   97/100     NonOverlappingTemplate
 22  15  10  12  14   7   7   4   4   5 0.000375     97/100     NonOverlappingTemplate
 32  17   6   9   4   4  11   7   2   8 0.000000 *   93/100  *  NonOverlappingTemplate
 31  15  12  10   9   5   7   3   2   6 0.000000 *   94/100  *  NonOverlappingTemplate
 17  14  12  22   7   8   4   6   4   6 0.000134     97/100     NonOverlappingTemplate
 13   9  11   9  10  10  13   7  11   7 0.911413     97/100     NonOverlappingTemplate
 12   6  14  12   8   7  12   8  14   7 0.474986     96/100     NonOverlappingTemplate
 29  19  11   7   8   6   5   5   6   4 0.000000 *   99/100     NonOverlappingTemplate
 19  14  10   9   6  11   7   6   8  10 0.108791     98/100     NonOverlappingTemplate
 12  16   9  10  10  14  14  10   3   2 0.028817    100/100     NonOverlappingTemplate
 17  18   9  11  10   8   5   9   4   9 0.032923     98/100     NonOverlappingTemplate
 21  15  11  12   9  10  11   6   4   1 0.000757     92/100  *  NonOverlappingTemplate
  9   8   9   7  13  11  11  10   8  14 0.867692    100/100     NonOverlappingTemplate
 13  10   5   9   7  10  12  10   9  15 0.595549     99/100     NonOverlappingTemplate
  6  12  12  13  10  11  12   7   9   8 0.816537    100/100     NonOverlappingTemplate
  8  12  12  15  12   8  10   6   8   9 0.678686    100/100     NonOverlappingTemplate
 13   6  10  13   9   7   9  10  13  10 0.798139    100/100     NonOverlappingTemplate
 10  11  11  17   8   6  11  11   7   8 0.474986    100/100     NonOverlappingTemplate
 10  12  12  11  18   5   9   8   5  10 0.171867     99/100     NonOverlappingTemplate
  9  10   6  17  10  12   9   7  10  10 0.534146     98/100     NonOverlappingTemplate
 14   8  10  11  10  13   9  11   6   8 0.816537     99/100     NonOverlappingTemplate
 13   8  14   7  10   8  11  12  13   4 0.419021     97/100     NonOverlappingTemplate
 11  11  12   8   8   6  12  13   8  11 0.851383     99/100     NonOverlappingTemplate
  9   9  15  13   8  13   8  14   5   6 0.275709     97/100     NonOverlappingTemplate
 13  14  13   8   5   9   9  10   9  10 0.678686     98/100     NonOverlappingTemplate
 19  10  14  16  11   6   7   7   3   7 0.007160     94/100  *  NonOverlappingTemplate
 14   6   8  13  15  11  11  12   5   5 0.181557     99/100     NonOverlappingTemplate
```

18	17	11	3	12	6	8	11	9	5	0.010988		97/100		NonOverlappingTemplate
7	7	10	12	13	10	13	10	8	10	0.883171		100/100		NonOverlappingTemplate
10	11	9	11	15	13	9	8	6	8	0.719747		98/100		NonOverlappingTemplate
9	11	12	13	9	7	9	11	8	11	0.955835		98/100		NonOverlappingTemplate
5	7	8	11	8	10	13	14	15	9	0.401199		100/100		NonOverlappingTemplate
20	15	8	5	11	10	10	10	6	5	0.020548		97/100		NonOverlappingTemplate
18	10	9	10	11	9	7	10	9	7	0.474986		97/100		NonOverlappingTemplate
14	8	10	12	11	14	10	9	7	5	0.574903		100/100		NonOverlappingTemplate
23	7	9	7	7	12	9	10	7	9	0.011791		100/100		NonOverlappingTemplate
13	11	9	11	10	6	11	12	3	14	0.366918		99/100		NonOverlappingTemplate
9	6	15	7	10	10	7	12	13	11	0.595549		98/100		NonOverlappingTemplate
6	8	12	15	10	7	12	7	9	14	0.455937		99/100		NonOverlappingTemplate
20	13	6	13	9	10	5	11	11	2	0.007160		95/100	*	NonOverlappingTemplate
18	15	8	12	10	4	9	5	13	6	0.030806		100/100		NonOverlappingTemplate
12	12	18	10	14	10	7	5	7	5	0.075719		99/100		NonOverlappingTemplate
12	10	12	7	10	12	4	15	9	9	0.494392		97/100		NonOverlappingTemplate
7	12	15	10	7	9	14	7	11	8	0.554420		98/100		NonOverlappingTemplate
13	8	13	13	7	8	7	16	7	8	0.334538		99/100		NonOverlappingTemplate
12	13	6	10	9	15	11	13	3	8	0.224821		97/100		NonOverlappingTemplate
13	5	11	12	15	5	8	15	4	12	0.071177		100/100		NonOverlappingTemplate
18	10	12	9	8	8	9	12	8	6	0.334538		95/100	*	NonOverlappingTemplate
14	5	14	12	6	11	12	7	8	11	0.383827		99/100		NonOverlappingTemplate
18	5	10	7	8	5	12	9	13	13	0.090936		97/100		NonOverlappingTemplate
12	11	13	12	11	10	7	10	7	7	0.867692		100/100		NonOverlappingTemplate
11	10	13	13	7	12	10	9	8	7	0.867692		100/100		NonOverlappingTemplate
15	14	12	12	2	7	12	8	11	7	0.122325		98/100		NonOverlappingTemplate
18	15	14	10	13	6	8	5	7	4	0.015598		97/100		NonOverlappingTemplate
11	18	9	12	7	6	8	11	13	5	0.145326		100/100		NonOverlappingTemplate
9	11	13	14	10	5	11	10	8	9	0.759756		99/100		NonOverlappingTemplate
10	17	10	3	6	14	9	14	10	7	0.075719		99/100		NonOverlappingTemplate
31	16	8	10	8	4	6	8	3	6	0.000000	*	90/100	*	NonOverlappingTemplate
18	15	15	5	12	10	7	5	6	7	0.016717		93/100	*	NonOverlappingTemplate
11	8	16	15	7	9	6	9	6	13	0.224821		100/100		NonOverlappingTemplate
24	16	9	9	11	9	5	9	4	4	0.000114		95/100	*	NonOverlappingTemplate
27	16	9	13	11	5	7	3	3	6	0.000000	*	97/100		NonOverlappingTemplate
18	18	8	9	9	13	5	9	4	7	0.010988		98/100		NonOverlappingTemplate
6	6	10	10	16	8	8	15	9	12	0.304126		99/100		NonOverlappingTemplate
9	11	11	14	9	12	12	5	8	9	0.759756		100/100		NonOverlappingTemplate
23	17	18	5	10	7	7	7	4	2	0.000002	*	91/100	*	NonOverlappingTemplate
21	22	8	10	7	9	7	2	7	7	0.000026	*	96/100		NonOverlappingTemplate
21	13	12	14	6	5	7	5	7	10	0.005358		92/100	*	NonOverlappingTemplate
11	15	14	15	6	4	13	9	6	7	0.080519		100/100		NonOverlappingTemplate
68	16	6	5	1	1	3	0	0	0	0.000000	*	59/100	*	NonOverlappingTemplate
30	18	14	15	6	2	5	5	4	1	0.000000	*	89/100	*	NonOverlappingTemplate
32	13	12	14	9	3	8	3	4	2	0.000000	*	94/100	*	NonOverlappingTemplate
11	11	11	9	13	10	8	8	9	10	0.987896		99/100		NonOverlappingTemplate
19	12	12	11	12	9	11	5	7	2	0.021999		98/100		NonOverlappingTemplate
15	10	9	10	12	9	11	6	8	10	0.816537		98/100		NonOverlappingTemplate
35	15	12	16	7	4	3	1	4	3	0.000000	*	97/100		NonOverlappingTemplate
10	11	6	11	16	11	8	9	11	7	0.637119		99/100		NonOverlappingTemplate

9	15	7	5	10	14	5	15	8	12	0.145326		100/100		NonOverlappingTemplate
12	9	11	11	6	4	12	15	14	6	0.213309		98/100		NonOverlappingTemplate
7	13	12	7	14	12	7	10	12	6	0.534146		99/100		NonOverlappingTemplate
18	8	9	11	4	10	11	13	8	8	0.191687		95/100	*	NonOverlappingTemplate
6	8	8	13	9	13	9	8	14	12	0.657933		100/100		NonOverlappingTemplate
15	11	10	6	11	14	8	6	13	6	0.319084		96/100		NonOverlappingTemplate
17	10	14	17	12	4	11	6	2	7	0.003712		100/100		NonOverlappingTemplate
11	11	7	14	12	6	8	12	9	10	0.779188		99/100		NonOverlappingTemplate
3	14	6	12	13	13	11	13	9	6	0.162606		100/100		NonOverlappingTemplate
19	16	9	10	11	11	7	5	8	4	0.021999		95/100	*	NonOverlappingTemplate
9	10	10	6	16	3	11	10	12	13	0.236810		100/100		NonOverlappingTemplate
19	8	10	9	12	8	6	10	9	9	0.262249		95/100	*	NonOverlappingTemplate
6	7	11	8	16	14	8	12	6	12	0.275709		100/100		NonOverlappingTemplate
13	12	9	13	9	9	12	4	8	11	0.637119		98/100		NonOverlappingTemplate
63	10	11	4	5	1	0	2	3	1	0.000000	*	77/100	*	NonOverlappingTemplate
21	13	10	14	13	5	5	9	5	5	0.002374		98/100		NonOverlappingTemplate
17	19	7	8	8	11	10	7	8	5	0.028817		97/100		NonOverlappingTemplate
20	13	9	12	11	13	6	5	5	6	0.014550		94/100	*	NonOverlappingTemplate
9	12	8	7	8	8	7	18	13	10	0.289667		100/100		NonOverlappingTemplate
21	15	8	13	8	8	13	8	4	2	0.000954		99/100		NonOverlappingTemplate
8	11	11	4	11	14	13	5	9	14	0.275709		100/100		NonOverlappingTemplate
10	14	11	10	5	8	12	8	10	12	0.759756		98/100		NonOverlappingTemplate
17	10	12	6	11	8	7	8	9	12	0.419021		98/100		NonOverlappingTemplate
10	14	12	8	8	12	8	6	10	12	0.779188		97/100		NonOverlappingTemplate
12	7	18	13	14	11	9	5	3	8	0.032923		99/100		NonOverlappingTemplate
10	13	11	10	7	7	7	12	10	13	0.834308		97/100		NonOverlappingTemplate
11	4	16	15	11	4	12	11	8	8	0.096578		100/100		NonOverlappingTemplate
16	14	8	13	12	1	11	11	6	8	0.045675		99/100		NonOverlappingTemplate
25	17	7	8	8	10	9	7	5	4	0.000037	*	89/100	*	NonOverlappingTemplate
17	13	8	11	8	10	7	11	6	9	0.401199		98/100		NonOverlappingTemplate
13	15	8	15	11	10	7	9	5	7	0.289667		99/100		NonOverlappingTemplate
20	7	10	12	8	10	7	9	8	9	0.153763		97/100		NonOverlappingTemplate
12	11	11	7	13	6	10	8	10	12	0.851383		100/100		NonOverlappingTemplate
9	11	7	8	6	14	13	10	9	13	0.678686		100/100		NonOverlappingTemplate
23	19	11	8	10	7	8	5	5	4	0.000051	*	94/100	*	NonOverlappingTemplate
5	14	6	13	11	14	6	13	13	5	0.115387		100/100		NonOverlappingTemplate
51	13	9	9	7	2	2	3	4	0	0.000000	*	88/100	*	NonOverlappingTemplate
13	13	15	6	7	10	10	9	7	10	0.554420		99/100		NonOverlappingTemplate
14	11	7	15	14	7	8	7	7	10	0.366918		95/100	*	NonOverlappingTemplate
6	11	9	15	4	7	14	17	11	6	0.048716		100/100		NonOverlappingTemplate
11	13	6	18	10	12	7	7	6	10	0.171867		98/100		NonOverlappingTemplate
19	13	16	10	4	5	8	9	7	9	0.016717		95/100	*	NonOverlappingTemplate
12	13	8	8	8	9	15	9	9	9	0.798139		99/100		NonOverlappingTemplate
8	7	9	13	13	16	11	8	11	4	0.275709		99/100		NonOverlappingTemplate
14	12	14	11	10	6	8	6	11	8	0.554420		99/100		NonOverlappingTemplate
11	15	12	5	11	9	13	7	12	5	0.319084		99/100		NonOverlappingTemplate
34	9	12	8	6	7	7	7	5	5	0.000000	*	95/100	*	NonOverlappingTemplate
5	14	15	11	7	8	11	12	8	9	0.437274		100/100		NonOverlappingTemplate
17	18	12	10	5	6	5	16	8	3	0.001296		98/100		NonOverlappingTemplate
9	7	15	11	8	13	11	11	7	8	0.699313		100/100		NonOverlappingTemplate

29	15	8	10	7	8	4	7	5	7	0.000000	*	91/100	* NonOverlappingTemplate
8	9	11	11	11	9	15	6	8	12	0.759756		100/100	NonOverlappingTemplate
9	13	9	11	9	5	12	12	11	9	0.851383		99/100	NonOverlappingTemplate
10	14	11	7	16	10	6	10	12	4	0.224821		96/100	NonOverlappingTemplate
9	10	10	5	12	9	10	8	15	12	0.699313		99/100	NonOverlappingTemplate
28	14	10	10	6	5	6	7	7	7	0.000003	*	93/100	* NonOverlappingTemplate
8	12	13	8	13	8	12	8	9	9	0.883171		100/100	NonOverlappingTemplate
17	20	15	7	10	9	3	11	6	2	0.000253		95/100	* NonOverlappingTemplate
14	16	12	7	9	10	13	7	5	7	0.224821		99/100	NonOverlappingTemplate
7	14	10	8	11	7	13	11	11	8	0.798139		100/100	NonOverlappingTemplate
13	13	16	8	10	5	6	7	11	11	0.275709		99/100	NonOverlappingTemplate
20	17	7	12	10	7	7	4	11	5	0.003996		99/100	NonOverlappingTemplate
8	13	3	11	8	12	12	7	17	9	0.145326		100/100	NonOverlappingTemplate
15	9	14	6	18	10	7	7	7	7	0.071177		96/100	NonOverlappingTemplate
6	10	11	9	14	8	15	11	10	6	0.534146		97/100	NonOverlappingTemplate
11	15	14	15	6	4	13	8	7	7	0.090936		100/100	NonOverlappingTemplate
17	11	7	15	13	13	5	4	7	8	0.040108		98/100	OverlappingTemplate
18	16	9	8	7	10	7	7	12	6	0.085587		96/100	Universal
70	9	8	3	2	4	2	1	1	0	0.000000	*	70/100	* ApproximateEntropy
0	1	0	0	1	0	0	0	1	1	----		4/4	RandomExcursions
0	1	0	0	0	0	0	0	1	2	----		4/4	RandomExcursions
1	0	0	1	1	0	0	0	0	1	----		4/4	RandomExcursions
0	0	0	0	0	1	0	0	2	1	----		4/4	RandomExcursions
0	0	0	2	2	0	0	0	0	0	----		4/4	RandomExcursions
0	1	1	1	0	1	0	0	0	0	----		4/4	RandomExcursions
2	0	0	0	0	1	0	0	0	1	----		4/4	RandomExcursions
1	0	1	1	0	0	0	0	1	0	----		4/4	RandomExcursions
1	0	1	0	0	1	0	0	0	1	----		4/4	RandomExcursionsVariant
1	0	1	0	0	1	0	0	0	1	----		4/4	RandomExcursionsVariant
1	0	1	0	0	0	0	0	0	2	----		4/4	RandomExcursionsVariant
1	1	0	0	1	0	1	0	0	0	----		4/4	RandomExcursionsVariant
1	1	1	0	1	0	0	0	0	0	----		4/4	RandomExcursionsVariant
1	1	1	0	1	0	0	0	0	0	----		4/4	RandomExcursionsVariant
0	1	1	0	0	1	0	1	0	0	----		4/4	RandomExcursionsVariant
0	0	0	1	0	0	1	0	2	0	----		4/4	RandomExcursionsVariant
0	0	0	1	1	0	0	1	0	1	----		4/4	RandomExcursionsVariant
1	0	2	0	1	0	0	0	0	0	----		4/4	RandomExcursionsVariant
1	1	0	2	0	0	0	0	0	0	----		4/4	RandomExcursionsVariant
1	0	1	1	0	0	0	0	1	0	----		4/4	RandomExcursionsVariant
1	0	0	1	1	0	0	0	0	1	----		4/4	RandomExcursionsVariant
1	0	0	0	1	1	0	0	1	0	----		4/4	RandomExcursionsVariant
1	0	0	0	0	0	2	0	1	0	----		4/4	RandomExcursionsVariant
1	0	0	0	0	1	1	0	0	1	----		4/4	RandomExcursionsVariant
1	0	0	1	0	1	0	0	0	1	----		4/4	RandomExcursionsVariant
1	0	0	0	2	0	1	0	0	0	----		4/4	RandomExcursionsVariant
18	11	12	13	9	10	6	6	8	7	0.191687		96/100	Serial
12	5	8	9	9	17	10	11	10	9	0.474986		97/100	Serial
10	11	14	11	11	10	10	10	6	7	0.883171		100/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 3 for a sample size = 4 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.3: Risultati del test NIST STS 2.1 su una sequenza di $100 \times 10^6\,\mathrm{bit}$ generati dal TRNG su mappa M_{ADC} a 4 rami, come mostrato in Fig. 5.8. I 2 bit prodotti ad ogni iterazione della mappa vengono memorizzati e poi inviati senza effettuare alcun post-processing. Differentemente dal caso A, in questo caso viene impiegato lo schema di codifica alternativo 01-00-10-11-01. La sequenza analizzata è prelevata in ingresso all'entropy distiller di Linux™.

A.7 Risultati test statistici NIST - Caso D

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <device_raw_N3K4F1_alt_code_pool.txt>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
8	11	11	11	7	13	8	10	13	8	0.897763	100/100	Frequency
12	15	4	9	11	8	13	11	7	10	0.437274	99/100	BlockFrequency
7	11	15	11	13	10	5	7	9	12	0.494392	100/100	CumulativeSums
10	9	10	11	7	15	11	6	14	7	0.554420	100/100	CumulativeSums
5	14	14	14	2	13	10	9	11	8	0.085587	99/100	Runs
12	8	6	9	11	11	13	7	11	12	0.834308	98/100	LongestRun
7	6	9	7	12	16	14	8	9	12	0.350485	99/100	Rank
16	10	14	18	9	5	7	9	5	7	0.028817	97/100	FFT
12	10	12	10	8	14	9	12	5	8	0.719747	99/100	NonOverlappingTemplate
8	16	11	9	8	9	13	12	6	8	0.534146	99/100	NonOverlappingTemplate
9	9	7	8	14	8	11	10	5	19	0.115387	100/100	NonOverlappingTemplate
10	11	14	7	10	10	12	10	8	8	0.924076	99/100	NonOverlappingTemplate
9	11	10	10	10	12	11	9	9	9	0.999438	99/100	NonOverlappingTemplate
6	9	11	9	6	9	8	13	13	16	0.401199	100/100	NonOverlappingTemplate
9	7	14	11	10	8	15	11	9	6	0.595549	100/100	NonOverlappingTemplate
6	15	10	8	11	7	11	14	10	8	0.574903	100/100	NonOverlappingTemplate
16	6	9	7	7	7	13	13	11	11	0.350485	98/100	NonOverlappingTemplate
9	9	15	9	9	15	9	8	10	7	0.657933	99/100	NonOverlappingTemplate
7	11	9	11	17	10	7	8	11	9	0.574903	100/100	NonOverlappingTemplate
12	5	8	10	4	7	12	15	9	18	0.045675	100/100	NonOverlappingTemplate
10	16	9	9	11	8	5	14	10	8	0.455937	99/100	NonOverlappingTemplate
11	13	12	10	10	14	13	6	6	5	0.383827	100/100	NonOverlappingTemplate
13	9	10	9	10	10	10	9	6	14	0.883171	100/100	NonOverlappingTemplate
9	7	11	15	10	8	8	9	11	12	0.834308	99/100	NonOverlappingTemplate
9	12	7	12	12	8	4	11	17	8	0.236810	100/100	NonOverlappingTemplate
14	11	13	9	9	11	8	10	10	5	0.759756	100/100	NonOverlappingTemplate
6	8	10	12	17	7	11	10	14	5	0.191687	99/100	NonOverlappingTemplate
13	10	12	11	10	9	7	11	8	9	0.964295	99/100	NonOverlappingTemplate
7	9	8	13	11	12	12	10	10	8	0.935716	100/100	NonOverlappingTemplate
6	12	12	8	6	12	12	11	9	12	0.759756	99/100	NonOverlappingTemplate
9	10	10	13	12	12	10	12	6	6	0.798139	100/100	NonOverlappingTemplate
10	15	12	11	11	10	9	4	8	10	0.616305	98/100	NonOverlappingTemplate
14	9	8	8	11	10	8	10	10	12	0.946308	97/100	NonOverlappingTemplate
11	10	8	11	9	10	11	8	10	12	0.996335	100/100	NonOverlappingTemplate
9	9	9	9	10	11	15	10	10	8	0.946308	100/100	NonOverlappingTemplate
14	7	8	6	12	10	14	8	12	9	0.595549	98/100	NonOverlappingTemplate
11	8	9	11	13	9	10	12	7	10	0.964295	100/100	NonOverlappingTemplate
11	18	14	9	4	8	11	8	6	11	0.108791	98/100	NonOverlappingTemplate
13	9	11	13	7	8	8	8	9	14	0.759756	99/100	NonOverlappingTemplate
11	10	10	5	16	14	8	12	8	6	0.304126	99/100	NonOverlappingTemplate

7	15	13	8	8	11	8	12	10	8	0.699313	99/100		NonOverlappingTemplate
10	9	11	15	13	10	13	6	7	6	0.474986	99/100		NonOverlappingTemplate
5	8	12	11	8	14	17	7	8	10	0.236810	100/100		NonOverlappingTemplate
7	7	12	9	13	8	13	10	11	10	0.867692	99/100		NonOverlappingTemplate
8	12	12	5	13	12	9	9	10	10	0.816537	100/100		NonOverlappingTemplate
6	9	12	9	12	9	9	7	17	10	0.474986	100/100		NonOverlappingTemplate
14	10	4	10	7	10	14	13	10	8	0.437274	97/100		NonOverlappingTemplate
12	12	8	9	12	11	8	9	9	10	0.983453	99/100		NonOverlappingTemplate
6	9	15	14	6	10	9	10	11	10	0.574903	100/100		NonOverlappingTemplate
10	5	13	12	14	13	11	5	11	6	0.304126	100/100		NonOverlappingTemplate
15	12	8	8	9	7	10	10	11	10	0.851383	97/100		NonOverlappingTemplate
7	12	6	20	7	6	14	9	9	10	0.045675	100/100		NonOverlappingTemplate
7	13	10	11	5	11	12	14	9	8	0.637119	98/100		NonOverlappingTemplate
5	7	8	12	8	11	14	11	10	14	0.534146	100/100		NonOverlappingTemplate
9	9	16	14	9	9	12	7	7	8	0.514124	98/100		NonOverlappingTemplate
9	11	14	13	11	10	4	10	7	11	0.595549	99/100		NonOverlappingTemplate
12	8	13	15	6	5	14	7	6	14	0.122325	99/100		NonOverlappingTemplate
8	6	8	13	12	11	9	12	13	8	0.779188	100/100		NonOverlappingTemplate
11	8	9	6	8	11	13	13	14	7	0.637119	97/100		NonOverlappingTemplate
5	11	9	16	10	13	6	9	11	10	0.437274	99/100		NonOverlappingTemplate
12	5	13	8	9	13	8	12	13	7	0.554420	100/100		NonOverlappingTemplate
9	15	16	10	8	8	8	12	8	6	0.366918	99/100		NonOverlappingTemplate
11	12	9	9	5	14	9	13	10	8	0.719747	98/100		NonOverlappingTemplate
9	9	11	7	10	10	14	14	9	7	0.798139	99/100		NonOverlappingTemplate
8	16	4	7	12	10	6	10	11	16	0.115387	97/100		NonOverlappingTemplate
14	8	6	12	10	9	8	11	12	10	0.834308	97/100		NonOverlappingTemplate
11	6	7	14	11	14	9	9	11	8	0.678686	98/100		NonOverlappingTemplate
11	8	7	12	8	12	11	16	6	9	0.534146	98/100		NonOverlappingTemplate
6	9	13	15	11	15	13	6	7	5	0.137282	99/100		NonOverlappingTemplate
12	4	11	11	12	9	12	10	6	13	0.574903	99/100		NonOverlappingTemplate
6	10	8	11	9	9	13	11	13	10	0.897763	99/100		NonOverlappingTemplate
10	13	8	16	9	10	9	11	9	5	0.554420	100/100		NonOverlappingTemplate
9	10	13	12	12	10	13	7	10	4	0.616305	99/100		NonOverlappingTemplate
16	8	13	11	10	10	9	10	5	8	0.534146	100/100		NonOverlappingTemplate
7	14	10	12	13	11	9	9	9	6	0.759756	100/100		NonOverlappingTemplate
9	11	10	5	10	11	11	11	7	15	0.699313	99/100		NonOverlappingTemplate
6	9	11	10	16	10	9	12	6	11	0.574903	99/100		NonOverlappingTemplate
14	12	11	7	9	11	8	6	11	11	0.798139	99/100		NonOverlappingTemplate
5	5	5	14	15	6	16	14	10	10	0.030806	98/100		NonOverlappingTemplate
10	9	12	5	11	7	11	11	14	10	0.759756	99/100		NonOverlappingTemplate
6	10	10	8	9	11	13	10	10	13	0.911413	100/100		NonOverlappingTemplate
7	12	7	10	13	8	11	12	7	13	0.759756	99/100		NonOverlappingTemplate
12	10	12	10	8	14	9	12	5	8	0.719747	99/100		NonOverlappingTemplate
7	13	7	7	9	12	14	13	11	7	0.574903	100/100		NonOverlappingTemplate
13	6	7	9	13	9	11	6	13	13	0.534146	100/100		NonOverlappingTemplate
11	11	8	8	15	11	8	11	4	13	0.474986	100/100		NonOverlappingTemplate
6	13	9	16	10	9	13	7	9	8	0.474986	100/100		NonOverlappingTemplate
6	6	13	14	7	17	11	8	13	5	0.080519	99/100		NonOverlappingTemplate
15	9	9	9	10	8	6	9	10	15	0.595549	94/100	*	NonOverlappingTemplate
12	14	3	13	5	11	12	10	13	7	0.181557	100/100		NonOverlappingTemplate

11	9	8	13	11	12	10	12	10	4	0.739918	99/100	NonOverlappingTemplate
7	12	12	10	6	7	14	9	12	11	0.699313	100/100	NonOverlappingTemplate
11	6	6	17	6	8	12	11	8	15	0.137282	98/100	NonOverlappingTemplate
5	12	13	17	11	7	14	5	4	12	0.037566	99/100	NonOverlappingTemplate
15	17	8	9	9	10	8	7	9	8	0.366918	99/100	NonOverlappingTemplate
11	9	12	11	3	8	16	11	7	12	0.275709	100/100	NonOverlappingTemplate
12	5	11	15	12	9	10	9	11	6	0.554420	98/100	NonOverlappingTemplate
9	10	10	6	17	8	11	10	7	12	0.494392	100/100	NonOverlappingTemplate
15	14	8	9	8	10	6	12	10	8	0.595549	100/100	NonOverlappingTemplate
7	8	11	6	11	14	9	15	8	11	0.554420	100/100	NonOverlappingTemplate
14	10	9	15	11	9	12	7	6	7	0.514124	98/100	NonOverlappingTemplate
10	9	9	2	8	9	14	13	13	13	0.249284	100/100	NonOverlappingTemplate
13	8	6	6	9	11	17	14	4	12	0.085587	100/100	NonOverlappingTemplate
11	18	10	8	6	4	12	8	12	11	0.145326	98/100	NonOverlappingTemplate
14	12	9	10	7	5	6	11	11	15	0.366918	100/100	NonOverlappingTemplate
12	10	5	14	9	11	11	10	7	11	0.759756	98/100	NonOverlappingTemplate
9	8	19	5	3	11	9	11	15	10	0.026948	97/100	NonOverlappingTemplate
7	11	5	11	16	15	10	11	7	7	0.236810	99/100	NonOverlappingTemplate
13	13	9	13	7	12	10	2	9	12	0.275709	100/100	NonOverlappingTemplate
12	6	7	16	13	6	17	6	9	8	0.066882	99/100	NonOverlappingTemplate
6	12	16	11	7	11	9	8	7	13	0.437274	100/100	NonOverlappingTemplate
12	14	4	8	4	11	18	10	7	12	0.042808	98/100	NonOverlappingTemplate
7	13	10	10	12	13	7	11	5	12	0.637119	100/100	NonOverlappingTemplate
15	7	9	13	12	6	8	15	3	12	0.102526	97/100	NonOverlappingTemplate
11	11	14	11	3	9	10	9	9	13	0.534146	100/100	NonOverlappingTemplate
7	10	14	16	13	7	15	7	4	7	0.071177	100/100	NonOverlappingTemplate
13	6	14	11	13	10	10	6	7	10	0.574903	99/100	NonOverlappingTemplate
15	8	9	9	9	12	7	9	15	7	0.534146	98/100	NonOverlappingTemplate
15	16	6	7	10	8	9	8	10	11	0.383827	100/100	NonOverlappingTemplate
13	7	10	6	10	15	11	11	5	12	0.437274	97/100	NonOverlappingTemplate
11	13	10	11	14	7	12	6	7	9	0.678686	100/100	NonOverlappingTemplate
12	8	9	7	8	13	10	11	12	10	0.935716	99/100	NonOverlappingTemplate
14	12	5	13	4	8	14	12	11	7	0.191687	97/100	NonOverlappingTemplate
14	7	15	8	8	12	13	9	7	7	0.437274	100/100	NonOverlappingTemplate
7	15	7	11	8	7	3	12	12	18	0.037566	100/100	NonOverlappingTemplate
13	11	12	14	10	9	9	4	10	8	0.616305	97/100	NonOverlappingTemplate
11	7	16	10	11	10	10	12	7	6	0.574903	99/100	NonOverlappingTemplate
10	13	12	6	10	12	12	6	8	11	0.759756	98/100	NonOverlappingTemplate
10	13	10	10	14	14	10	6	7	6	0.514124	99/100	NonOverlappingTemplate
4	9	11	9	14	10	12	7	15	9	0.401199	100/100	NonOverlappingTemplate
11	8	5	18	4	19	11	8	12	4	0.002374	100/100	NonOverlappingTemplate
12	13	11	9	15	13	5	7	5	10	0.289667	98/100	NonOverlappingTemplate
11	10	5	13	7	11	11	9	15	8	0.574903	99/100	NonOverlappingTemplate
8	10	12	14	10	6	10	13	9	8	0.798139	98/100	NonOverlappingTemplate
9	10	15	11	7	10	8	9	11	10	0.897763	99/100	NonOverlappingTemplate
7	13	13	8	6	15	12	5	10	11	0.334538	98/100	NonOverlappingTemplate
11	11	10	9	11	10	5	12	11	10	0.946308	96/100	NonOverlappingTemplate
11	10	9	9	10	15	10	5	11	10	0.798139	98/100	NonOverlappingTemplate
14	10	11	10	6	13	4	10	8	14	0.366918	99/100	NonOverlappingTemplate
8	8	10	12	11	13	16	7	9	6	0.494392	99/100	NonOverlappingTemplate

5	10	12	14	5	14	12	10	7	11	0.350485	99/100	NonOverlappingTemplate
10	15	7	10	8	7	9	13	12	9	0.719747	99/100	NonOverlappingTemplate
9	7	12	11	7	10	15	11	9	9	0.816537	98/100	NonOverlappingTemplate
12	12	12	8	11	10	6	5	11	13	0.657933	99/100	NonOverlappingTemplate
5	6	11	13	10	10	12	10	12	11	0.739918	99/100	NonOverlappingTemplate
11	17	12	9	6	8	8	13	9	7	0.366918	99/100	NonOverlappingTemplate
9	13	9	8	16	7	14	9	6	9	0.401199	99/100	NonOverlappingTemplate
8	15	8	5	13	15	10	10	6	10	0.289667	100/100	NonOverlappingTemplate
8	6	7	14	11	7	16	7	9	15	0.181557	100/100	NonOverlappingTemplate
6	12	12	9	12	10	12	10	11	6	0.834308	100/100	NonOverlappingTemplate
6	11	11	8	10	9	14	12	10	9	0.883171	99/100	NonOverlappingTemplate
10	12	15	8	15	14	7	8	6	5	0.171867	100/100	NonOverlappingTemplate
11	8	12	9	8	14	8	11	9	10	0.935716	98/100	NonOverlappingTemplate
5	6	13	16	10	10	7	13	6	14	0.137282	100/100	NonOverlappingTemplate
8	12	11	8	9	8	17	8	11	8	0.574903	98/100	NonOverlappingTemplate
7	12	7	11	12	8	11	12	7	13	0.798139	99/100	NonOverlappingTemplate
11	14	13	7	9	6	8	11	12	9	0.719747	99/100	OverlappingTemplate
10	11	7	12	7	12	9	10	8	14	0.851383	98/100	Universal
11	5	13	9	12	12	5	10	16	7	0.249284	99/100	ApproximateEntropy
6	5	8	10	6	2	6	9	5	7	0.534146	63/64	RandomExcursions
9	4	8	8	4	6	9	9	4	3	0.407091	64/64	RandomExcursions
7	1	11	8	5	8	4	9	8	3	0.110952	62/64	RandomExcursions
8	5	4	10	5	12	5	5	6	4	0.253551	63/64	RandomExcursions
11	9	2	5	6	5	7	3	10	6	0.162606	64/64	RandomExcursions
9	8	3	4	9	10	5	6	4	6	0.407091	64/64	RandomExcursions
5	5	12	3	6	8	7	6	5	7	0.437274	63/64	RandomExcursions
3	6	4	7	11	4	6	6	11	6	0.253551	64/64	RandomExcursions
9	5	5	5	6	6	8	9	7	4	0.834308	63/64	RandomExcursionsVariant
8	6	6	3	12	6	5	3	7	8	0.299251	62/64	RandomExcursionsVariant
6	9	5	4	6	12	6	8	4	4	0.324180	62/64	RandomExcursionsVariant
5	9	7	4	6	6	7	7	6	7	0.964295	63/64	RandomExcursionsVariant
4	9	9	5	9	6	4	9	6	3	0.437274	63/64	RandomExcursionsVariant
6	6	11	12	7	6	5	7	4	0	0.043745	63/64	RandomExcursionsVariant
6	12	11	5	6	7	2	3	7	5	0.090936	62/64	RandomExcursionsVariant
5	12	8	6	5	8	5	8	4	3	0.299251	64/64	RandomExcursionsVariant
5	8	7	5	7	9	5	3	4	11	0.407091	64/64	RandomExcursionsVariant
6	9	4	10	9	5	4	6	6	5	0.602458	63/64	RandomExcursionsVariant
5	14	7	8	4	7	4	5	7	3	0.090936	64/64	RandomExcursionsVariant
7	6	5	6	10	6	9	4	6	5	0.804337	64/64	RandomExcursionsVariant
6	3	4	7	8	7	5	8	7	9	0.772760	64/64	RandomExcursionsVariant
5	5	3	8	4	11	3	7	10	8	0.195163	64/64	RandomExcursionsVariant
3	6	6	9	6	4	9	6	10	5	0.534146	64/64	RandomExcursionsVariant
4	4	7	9	5	6	8	2	10	9	0.299251	64/64	RandomExcursionsVariant
4	4	6	11	4	6	4	7	9	9	0.350485	64/64	RandomExcursionsVariant
5	5	7	5	6	8	3	7	7	11	0.602458	64/64	RandomExcursionsVariant
8	8	10	10	16	8	10	7	12	11	0.719747	99/100	Serial
9	10	6	6	13	9	10	13	14	10	0.657933	100/100	Serial
10	6	10	8	9	15	7	12	14	9	0.574903	98/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 60 for a sample size = 64 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.4: Risultati del test NIST STS 2.1 su una sequenza di 100×10^6 bit prelevati in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso C.

A.8 Risultati test statistici NIST - Caso E

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <device_neumann_N3K4F1_alt_code.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION		STATISTICAL TEST
10	8	14	8	11	10	13	7	8	11	0.851383	99/100		Frequency
21	17	9	10	10	6	12	7	4	4	0.001296	98/100		BlockFrequency
11	8	8	13	7	9	9	9	13	13	0.851383	100/100		CumulativeSums
11	7	13	7	12	8	12	11	9	10	0.897763	100/100		CumulativeSums
77	11	2	2	1	1	1	2	1	2	0.000000 *	56/100	*	Runs
9	7	8	11	7	14	4	24	7	9	0.000883	99/100		LongestRun
10	7	8	10	13	7	12	11	12	10	0.911413	98/100		Rank
13	6	16	7	13	10	6	8	10	11	0.350485	98/100		FFT
16	12	14	15	7	7	9	7	7	6	0.145326	95/100	*	NonOverlappingTemplate
14	15	14	11	7	9	6	6	7	11	0.275709	99/100		NonOverlappingTemplate
9	9	9	16	11	9	4	7	12	14	0.304126	100/100		NonOverlappingTemplate
12	12	11	21	13	12	5	5	6	3	0.002203	99/100		NonOverlappingTemplate
11	12	10	11	10	9	4	9	13	11	0.798139	99/100		NonOverlappingTemplate
12	9	13	9	7	11	10	10	7	12	0.924076	100/100		NonOverlappingTemplate
12	19	6	4	12	8	7	12	14	6	0.025193	99/100		NonOverlappingTemplate
14	12	12	16	9	8	6	9	9	5	0.289667	98/100		NonOverlappingTemplate
7	9	12	8	8	10	11	11	11	13	0.946308	99/100		NonOverlappingTemplate
9	9	13	9	12	10	6	14	4	14	0.350485	100/100		NonOverlappingTemplate
9	12	12	11	10	4	14	9	12	7	0.574903	98/100		NonOverlappingTemplate
10	7	9	9	9	17	8	8	5	18	0.071177	99/100		NonOverlappingTemplate
7	13	12	9	5	8	11	10	13	12	0.678686	100/100		NonOverlappingTemplate
14	10	15	8	10	11	11	7	6	8	0.574903	100/100		NonOverlappingTemplate
15	12	11	6	9	11	8	7	9	12	0.678686	99/100		NonOverlappingTemplate
11	17	13	11	12	4	8	13	7	4	0.071177	97/100		NonOverlappingTemplate
10	12	4	6	8	8	17	13	10	12	0.181557	99/100		NonOverlappingTemplate
14	9	8	10	13	12	4	11	6	13	0.383827	98/100		NonOverlappingTemplate
10	10	8	13	7	10	16	7	9	10	0.657933	99/100		NonOverlappingTemplate
5	18	7	9	9	10	10	11	12	9	0.304126	100/100		NonOverlappingTemplate
15	7	7	14	9	10	12	9	10	7	0.595549	98/100		NonOverlappingTemplate
11	11	6	12	8	12	5	10	17	8	0.289667	98/100		NonOverlappingTemplate
8	9	9	9	10	10	8	10	13	14	0.935716	100/100		NonOverlappingTemplate
12	10	12	9	9	5	8	10	12	13	0.816537	100/100		NonOverlappingTemplate
11	8	10	10	13	13	10	8	8	9	0.955835	99/100		NonOverlappingTemplate
10	12	13	11	11	11	4	8	10	10	0.779188	100/100		NonOverlappingTemplate
7	11	11	15	4	13	13	9	9	8	0.383827	98/100		NonOverlappingTemplate
15	12	8	13	15	6	12	4	5	10	0.096578	98/100		NonOverlappingTemplate
5	9	4	15	7	13	11	17	8	11	0.066882	99/100		NonOverlappingTemplate
15	7	11	8	10	11	13	10	8	7	0.719747	99/100		NonOverlappingTemplate
11	10	9	12	6	9	9	11	13	10	0.946308	100/100		NonOverlappingTemplate
16	10	10	10	8	6	16	7	9	8	0.304126	99/100		NonOverlappingTemplate

14	10	9	13	4	9	11	8	9	13	0.554420	97/100		NonOverlappingTemplate
4	13	9	13	8	8	10	8	12	15	0.383827	100/100		NonOverlappingTemplate
9	6	13	11	12	7	11	15	8	8	0.595549	99/100		NonOverlappingTemplate
7	7	13	12	8	7	14	7	11	14	0.474986	100/100		NonOverlappingTemplate
13	6	10	10	12	10	12	11	7	9	0.883171	98/100		NonOverlappingTemplate
16	9	13	9	8	14	9	12	5	5	0.202268	98/100		NonOverlappingTemplate
12	14	8	10	11	14	12	4	9	6	0.366918	99/100		NonOverlappingTemplate
12	12	9	10	7	12	13	11	10	4	0.657933	99/100		NonOverlappingTemplate
12	14	17	10	9	9	7	7	9	6	0.304126	99/100		NonOverlappingTemplate
11	13	5	8	12	13	8	12	7	11	0.637119	99/100		NonOverlappingTemplate
12	9	11	6	8	11	11	12	10	10	0.955835	100/100		NonOverlappingTemplate
12	9	7	13	9	9	11	12	5	13	0.699313	100/100		NonOverlappingTemplate
8	9	7	11	9	3	11	14	14	14	0.249284	98/100		NonOverlappingTemplate
12	9	10	8	12	12	10	8	12	7	0.946308	98/100		NonOverlappingTemplate
7	11	8	12	11	10	9	11	10	11	0.987896	100/100		NonOverlappingTemplate
8	10	7	9	11	16	15	10	8	6	0.383827	99/100		NonOverlappingTemplate
16	8	11	8	9	11	11	6	9	11	0.678686	98/100		NonOverlappingTemplate
9	8	14	11	8	10	12	6	9	13	0.779188	98/100		NonOverlappingTemplate
13	8	10	13	9	8	13	7	12	7	0.759756	98/100		NonOverlappingTemplate
15	10	14	8	10	9	9	7	5	13	0.437274	97/100		NonOverlappingTemplate
8	7	7	14	12	13	11	9	14	5	0.401199	100/100		NonOverlappingTemplate
13	6	17	8	13	10	5	9	7	12	0.181557	100/100		NonOverlappingTemplate
13	9	11	16	10	10	5	12	9	5	0.334538	100/100		NonOverlappingTemplate
8	13	10	11	8	7	15	7	13	8	0.595549	100/100		NonOverlappingTemplate
8	5	13	12	14	8	12	11	11	6	0.494392	99/100		NonOverlappingTemplate
9	9	15	10	9	8	10	14	8	8	0.779188	99/100		NonOverlappingTemplate
5	15	11	8	16	7	10	5	14	9	0.115387	99/100		NonOverlappingTemplate
10	7	12	9	10	9	9	11	8	15	0.867692	99/100		NonOverlappingTemplate
11	8	9	15	10	7	8	13	7	12	0.678686	98/100		NonOverlappingTemplate
2	15	12	9	8	5	13	11	16	9	0.048716	99/100		NonOverlappingTemplate
7	14	12	11	11	7	9	12	7	10	0.798139	99/100		NonOverlappingTemplate
8	12	14	9	6	14	10	9	11	7	0.657933	100/100		NonOverlappingTemplate
11	11	5	8	6	9	10	18	9	13	0.202268	100/100		NonOverlappingTemplate
7	14	7	13	13	9	12	6	10	9	0.595549	99/100		NonOverlappingTemplate
10	8	10	12	9	14	8	10	12	7	0.897763	100/100		NonOverlappingTemplate
11	7	8	4	13	10	13	15	11	8	0.366918	99/100		NonOverlappingTemplate
14	9	11	7	11	4	11	12	13	8	0.514124	99/100		NonOverlappingTemplate
10	8	13	15	10	9	8	10	6	11	0.739918	99/100		NonOverlappingTemplate
6	8	9	12	8	9	17	11	7	13	0.366918	99/100		NonOverlappingTemplate
4	11	11	7	13	17	10	7	4	16	0.028817	100/100		NonOverlappingTemplate
10	11	3	12	9	17	11	9	14	4	0.071177	99/100		NonOverlappingTemplate
9	9	8	13	12	8	9	11	10	11	0.978072	100/100		NonOverlappingTemplate
16	12	14	15	7	7	9	7	7	6	0.145326	95/100	*	NonOverlappingTemplate
14	6	13	8	8	15	13	10	3	10	0.153763	99/100		NonOverlappingTemplate
5	13	8	12	14	11	9	9	15	4	0.202268	98/100		NonOverlappingTemplate
12	9	18	13	11	8	8	7	8	6	0.236810	98/100		NonOverlappingTemplate
6	10	12	14	12	5	10	10	15	6	0.304126	100/100		NonOverlappingTemplate
12	15	13	7	7	9	12	9	7	9	0.616305	99/100		NonOverlappingTemplate
14	9	10	12	11	7	11	6	9	11	0.834308	100/100		NonOverlappingTemplate
5	17	8	10	14	6	10	13	9	8	0.191687	100/100		NonOverlappingTemplate

9	11	11	12	7	11	16	7	9	7	0.616305	100/100	NonOverlappingTemplate
9	9	11	9	12	4	9	10	15	12	0.595549	98/100	NonOverlappingTemplate
13	12	14	10	10	8	7	5	11	10	0.657933	100/100	NonOverlappingTemplate
7	9	12	18	9	7	13	10	10	5	0.202268	100/100	NonOverlappingTemplate
11	10	16	8	11	8	11	8	9	8	0.779188	100/100	NonOverlappingTemplate
10	14	11	13	8	9	16	6	7	6	0.289667	100/100	NonOverlappingTemplate
9	8	8	6	8	16	13	8	15	9	0.319084	100/100	NonOverlappingTemplate
12	11	8	14	9	15	8	3	6	14	0.137282	98/100	NonOverlappingTemplate
12	13	10	10	10	13	14	7	5	6	0.455937	96/100	NonOverlappingTemplate
10	13	12	9	12	6	7	15	9	7	0.554420	100/100	NonOverlappingTemplate
4	13	13	8	11	13	8	9	11	10	0.595549	99/100	NonOverlappingTemplate
5	12	11	8	14	7	15	8	6	14	0.213309	100/100	NonOverlappingTemplate
9	5	11	9	12	12	6	12	9	15	0.514124	98/100	NonOverlappingTemplate
12	10	8	15	8	13	10	7	7	10	0.699313	100/100	NonOverlappingTemplate
12	6	15	16	5	11	9	10	8	8	0.236810	99/100	NonOverlappingTemplate
9	13	17	6	10	11	8	7	7	12	0.334538	100/100	NonOverlappingTemplate
11	11	10	13	9	12	6	10	10	8	0.935716	99/100	NonOverlappingTemplate
10	4	7	12	9	8	12	14	10	14	0.437274	98/100	NonOverlappingTemplate
14	16	10	9	8	5	11	9	12	6	0.319084	99/100	NonOverlappingTemplate
11	7	8	16	10	13	13	9	4	9	0.304126	97/100	NonOverlappingTemplate
12	14	9	12	7	9	9	6	12	10	0.779188	99/100	NonOverlappingTemplate
10	8	7	9	14	13	8	12	11	8	0.816537	100/100	NonOverlappingTemplate
10	13	9	7	7	6	8	18	8	14	0.153763	100/100	NonOverlappingTemplate
11	13	4	9	8	10	11	13	9	12	0.678686	100/100	NonOverlappingTemplate
13	10	6	13	11	7	11	7	8	14	0.595549	99/100	NonOverlappingTemplate
9	9	12	9	11	10	6	10	11	13	0.946308	99/100	NonOverlappingTemplate
12	16	7	15	7	8	5	9	15	6	0.080519	100/100	NonOverlappingTemplate
8	3	10	14	9	15	11	11	6	13	0.202268	100/100	NonOverlappingTemplate
10	11	7	15	11	8	10	14	7	7	0.595549	97/100	NonOverlappingTemplate
10	12	8	9	7	14	10	9	14	7	0.739918	100/100	NonOverlappingTemplate
5	7	10	11	10	8	14	7	14	14	0.383827	100/100	NonOverlappingTemplate
15	9	8	14	5	11	9	9	9	11	0.574903	98/100	NonOverlappingTemplate
9	12	6	9	12	10	14	9	12	7	0.779188	98/100	NonOverlappingTemplate
10	14	10	9	9	11	9	10	9	9	0.987896	99/100	NonOverlappingTemplate
19	8	8	4	7	13	12	6	11	12	0.051942	98/100	NonOverlappingTemplate
7	10	9	13	8	14	15	6	9	9	0.514124	97/100	NonOverlappingTemplate
14	11	10	11	10	11	8	11	8	6	0.883171	98/100	NonOverlappingTemplate
10	10	13	9	10	13	10	11	6	8	0.911413	99/100	NonOverlappingTemplate
9	8	5	15	12	7	12	9	13	10	0.514124	100/100	NonOverlappingTemplate
10	12	8	13	7	14	10	6	6	14	0.437274	100/100	NonOverlappingTemplate
8	8	13	9	14	7	6	13	5	17	0.115387	100/100	NonOverlappingTemplate
8	10	12	6	13	11	11	8	13	8	0.816537	100/100	NonOverlappingTemplate
9	9	9	13	6	14	10	13	6	11	0.637119	99/100	NonOverlappingTemplate
8	7	13	10	8	12	10	8	15	9	0.739918	100/100	NonOverlappingTemplate
13	15	6	5	7	9	11	12	11	11	0.419021	98/100	NonOverlappingTemplate
9	14	9	11	11	5	9	13	10	9	0.779188	97/100	NonOverlappingTemplate
6	6	12	11	10	13	13	13	7	9	0.595549	100/100	NonOverlappingTemplate
8	10	14	11	11	10	8	5	12	11	0.779188	100/100	NonOverlappingTemplate
13	13	11	12	6	19	8	7	7	4	0.037566	97/100	NonOverlappingTemplate
7	9	22	6	8	15	9	8	7	9	0.010988	99/100	NonOverlappingTemplate

8	13	15	8	7	7	15	11	7	9	0.383827	99/100	NonOverlappingTemplate
10	11	9	7	7	7	12	15	11	11	0.739918	99/100	NonOverlappingTemplate
13	4	10	6	10	11	2	11	13	20	0.004981	97/100	NonOverlappingTemplate
12	15	13	9	7	11	7	7	9	10	0.657933	99/100	NonOverlappingTemplate
6	15	6	9	9	5	14	8	13	15	0.129620	98/100	NonOverlappingTemplate
15	12	10	11	8	9	11	8	10	6	0.779188	98/100	NonOverlappingTemplate
7	16	11	7	7	7	12	5	10	18	0.055361	98/100	NonOverlappingTemplate
7	9	20	3	12	9	12	13	6	9	0.021999	99/100	NonOverlappingTemplate
12	6	9	10	10	14	9	9	11	10	0.911413	99/100	NonOverlappingTemplate
12	11	11	10	8	7	11	10	12	8	0.971699	100/100	NonOverlappingTemplate
7	8	9	11	8	10	11	7	13	16	0.595549	99/100	NonOverlappingTemplate
9	11	7	3	14	14	10	14	7	11	0.224821	99/100	NonOverlappingTemplate
7	15	9	13	10	8	10	8	11	9	0.798139	100/100	NonOverlappingTemplate
10	9	11	3	9	10	8	17	11	12	0.275709	99/100	NonOverlappingTemplate
10	4	16	10	10	10	11	9	15	5	0.191687	98/100	NonOverlappingTemplate
9	9	8	13	12	8	9	11	10	11	0.978072	100/100	NonOverlappingTemplate
12	17	10	8	9	11	10	8	11	4	0.350485	97/100	OverlappingTemplate
10	8	5	15	9	10	8	9	12	14	0.534146	99/100	Universal
12	8	8	12	14	11	13	8	10	4	0.514124	98/100	ApproximateEntropy
7	6	7	6	8	7	3	2	8	8	0.671779	62/62	RandomExcursions
4	6	8	11	5	5	2	6	8	7	0.407091	62/62	RandomExcursions
3	11	3	8	6	3	4	9	10	5	0.110952	61/62	RandomExcursions
4	7	5	9	5	2	9	9	5	7	0.468595	61/62	RandomExcursions
2	5	10	5	4	8	9	8	4	7	0.350485	61/62	RandomExcursions
5	6	6	3	7	5	7	9	7	7	0.911413	61/62	RandomExcursions
4	7	7	3	4	11	3	7	8	8	0.324180	62/62	RandomExcursions
9	8	3	3	5	8	6	8	4	8	0.534146	60/62	RandomExcursions
7	6	6	8	3	7	8	6	6	5	0.949602	62/62	RandomExcursionsVariant
6	8	8	5	6	5	8	3	8	5	0.862344	62/62	RandomExcursionsVariant
4	11	7	8	5	4	6	5	8	4	0.534146	62/62	RandomExcursionsVariant
5	8	12	6	8	6	3	5	6	3	0.299251	61/62	RandomExcursionsVariant
7	6	11	7	3	6	6	8	5	3	0.500934	61/62	RandomExcursionsVariant
8	1	15	5	4	6	1	7	8	7	0.003804	62/62	RandomExcursionsVariant
5	6	6	11	4	5	5	4	6	10	0.468595	62/62	RandomExcursionsVariant
5	6	7	5	3	10	4	8	8	6	0.671779	62/62	RandomExcursionsVariant
6	7	2	6	8	3	6	10	3	11	0.148094	62/62	RandomExcursionsVariant
5	8	5	3	11	5	2	9	6	8	0.232760	62/62	RandomExcursionsVariant
8	8	9	5	3	5	6	7	5	6	0.834308	62/62	RandomExcursionsVariant
9	7	11	5	5	5	5	4	6	5	0.602458	61/62	RandomExcursionsVariant
7	7	8	7	6	4	7	5	3	8	0.888137	62/62	RandomExcursionsVariant
6	8	6	7	6	8	5	3	8	5	0.911413	61/62	RandomExcursionsVariant
6	7	8	5	8	5	5	8	4	6	0.949602	60/62	RandomExcursionsVariant
8	6	11	5	3	7	2	6	7	7	0.378138	60/62	RandomExcursionsVariant
9	4	6	7	9	6	4	8	3	6	0.671779	59/62	RandomExcursionsVariant
6	6	9	4	6	9	6	8	5	3	0.739918	59/62	RandomExcursionsVariant
11	13	10	8	14	7	6	12	8	11	0.699313	98/100	Serial
9	10	6	15	6	15	6	9	9	15	0.181557	100/100	Serial
11	7	14	8	9	8	12	8	13	10	0.816537	97/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 59 for a sample size = 62 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.5: Risultati del test NIST STS 2.1 su una sequenza di $100 \times 10^6\,\mathrm{bit}$ generati dal TRNG sulla mappa M_{ADC} a 4 rami, mostrata in Fig. 5.8. I 2 bit prodotti ad ogni iterazione della mappa vengono memorizzati e successivamente processati per mezzo dell'algoritmo Von Neumann, mostrato in Tab. 3.1. Lo schema di codifica impiegato è quello alternativo, 01-00-10-11-01. La sequenza analizzata è prelevata in ingresso all'entropy distiller di Linux™.

A.9 Risultati test statistici NIST - Caso F

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <device_neumann_N3K4F1_alt_code_pool.txt>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
7	13	6	10	8	11	12	10	15	8	0.616305	100/100	Frequency
10	12	10	13	10	8	8	10	9	10	0.987896	100/100	BlockFrequency
6	13	8	6	14	11	9	6	14	13	0.319084	100/100	CumulativeSums
7	13	9	14	11	9	11	5	12	9	0.657933	100/100	CumulativeSums
8	8	7	10	9	5	15	14	14	10	0.350485	100/100	Runs
10	9	11	6	13	11	15	10	5	10	0.554420	99/100	LongestRun
5	8	10	10	16	10	13	14	7	7	0.289667	100/100	Rank
9	10	13	10	10	9	9	11	8	11	0.994250	97/100	FFT
7	8	14	10	7	10	10	11	9	14	0.779188	100/100	NonOverlappingTemplate
13	7	10	8	9	11	10	10	14	8	0.883171	99/100	NonOverlappingTemplate
9	10	11	10	8	10	10	10	9	13	0.996335	100/100	NonOverlappingTemplate
8	8	11	8	10	13	10	10	11	11	0.983453	100/100	NonOverlappingTemplate
12	12	13	12	11	10	6	6	8	10	0.759756	99/100	NonOverlappingTemplate
15	11	7	12	15	5	6	10	13	6	0.162606	98/100	NonOverlappingTemplate
9	6	12	4	9	13	17	11	11	8	0.202268	100/100	NonOverlappingTemplate
10	9	10	6	8	9	10	13	13	12	0.883171	100/100	NonOverlappingTemplate
12	11	11	10	12	13	6	11	6	8	0.779188	99/100	NonOverlappingTemplate
8	10	13	11	9	13	9	4	12	11	0.678686	98/100	NonOverlappingTemplate
4	11	12	11	10	12	11	8	10	11	0.816537	100/100	NonOverlappingTemplate
15	11	9	14	6	8	13	10	4	10	0.289667	97/100	NonOverlappingTemplate
9	10	7	10	12	3	13	15	12	9	0.334538	100/100	NonOverlappingTemplate
12	6	14	6	9	11	13	10	9	10	0.699313	99/100	NonOverlappingTemplate
17	9	12	5	12	10	6	9	11	9	0.334538	96/100	NonOverlappingTemplate
9	8	12	10	10	10	12	10	12	7	0.978072	100/100	NonOverlappingTemplate
11	7	9	15	13	8	12	14	2	9	0.145326	99/100	NonOverlappingTemplate
11	10	8	10	9	8	12	8	11	13	0.971699	98/100	NonOverlappingTemplate
8	9	8	13	15	8	6	10	11	12	0.657933	98/100	NonOverlappingTemplate
10	7	6	10	15	9	6	13	7	17	0.145326	96/100	NonOverlappingTemplate
10	8	7	9	14	9	8	9	16	10	0.616305	100/100	NonOverlappingTemplate
9	8	13	4	10	10	13	13	6	14	0.350485	98/100	NonOverlappingTemplate
13	10	8	9	12	13	10	5	13	7	0.637119	98/100	NonOverlappingTemplate
11	14	9	8	15	12	1	13	11	6	0.071177	100/100	NonOverlappingTemplate
6	6	6	16	12	15	10	8	11	10	0.224821	100/100	NonOverlappingTemplate
6	9	8	11	10	12	12	9	16	7	0.574903	97/100	NonOverlappingTemplate
8	15	17	9	4	10	10	8	12	7	0.153763	98/100	NonOverlappingTemplate
6	14	5	19	8	11	8	6	15	8	0.023545	99/100	NonOverlappingTemplate
8	6	16	9	5	16	8	11	15	6	0.058984	99/100	NonOverlappingTemplate
13	9	4	8	10	11	10	18	11	6	0.153763	100/100	NonOverlappingTemplate
14	7	12	10	5	14	11	6	12	9	0.419021	99/100	NonOverlappingTemplate
8	14	13	14	10	7	8	12	7	7	0.534146	100/100	NonOverlappingTemplate

11	9	13	8	15	5	8	14	6	11	0.334538	98/100	NonOverlappingTemplate
8	15	13	6	10	8	13	8	11	8	0.574903	100/100	NonOverlappingTemplate
13	5	8	10	15	11	6	11	10	11	0.514124	99/100	NonOverlappingTemplate
10	9	14	9	13	9	8	11	11	6	0.834308	100/100	NonOverlappingTemplate
7	14	10	9	9	14	8	12	8	9	0.779188	99/100	NonOverlappingTemplate
10	7	12	15	9	10	13	8	7	9	0.719747	99/100	NonOverlappingTemplate
9	8	4	12	13	11	17	11	8	7	0.224821	99/100	NonOverlappingTemplate
15	4	8	11	12	8	8	12	7	15	0.236810	100/100	NonOverlappingTemplate
10	12	7	12	8	13	13	14	3	8	0.289667	100/100	NonOverlappingTemplate
14	5	12	13	11	7	9	11	8	10	0.637119	100/100	NonOverlappingTemplate
12	13	11	6	12	10	5	10	9	12	0.699313	99/100	NonOverlappingTemplate
15	8	10	12	8	6	14	10	7	10	0.554420	99/100	NonOverlappingTemplate
10	10	11	8	11	7	10	13	10	10	0.983453	99/100	NonOverlappingTemplate
7	4	13	11	6	8	15	13	13	10	0.224821	99/100	NonOverlappingTemplate
14	13	11	6	7	10	7	14	11	7	0.474986	98/100	NonOverlappingTemplate
9	14	8	6	10	12	11	10	10	10	0.897763	99/100	NonOverlappingTemplate
13	6	9	13	14	11	6	8	8	12	0.534146	99/100	NonOverlappingTemplate
10	6	11	16	10	8	9	6	10	14	0.437274	98/100	NonOverlappingTemplate
12	7	15	8	6	7	8	10	10	17	0.213309	100/100	NonOverlappingTemplate
9	11	11	8	10	9	10	5	15	12	0.719747	99/100	NonOverlappingTemplate
9	13	8	13	10	10	8	15	8	6	0.616305	99/100	NonOverlappingTemplate
9	15	10	12	6	11	12	8	10	7	0.699313	99/100	NonOverlappingTemplate
11	8	9	14	4	12	15	8	7	12	0.319084	98/100	NonOverlappingTemplate
12	10	12	13	7	7	14	5	11	9	0.554420	98/100	NonOverlappingTemplate
7	11	13	12	8	13	9	11	9	7	0.851383	99/100	NonOverlappingTemplate
7	12	11	11	9	11	7	8	11	13	0.911413	99/100	NonOverlappingTemplate
12	8	8	11	7	11	13	10	10	10	0.955835	100/100	NonOverlappingTemplate
8	9	3	11	17	7	9	16	10	10	0.090936	99/100	NonOverlappingTemplate
10	12	11	15	4	8	7	11	12	10	0.494392	100/100	NonOverlappingTemplate
8	8	14	9	8	12	8	11	13	9	0.851383	99/100	NonOverlappingTemplate
8	6	7	9	10	14	12	11	10	13	0.739918	100/100	NonOverlappingTemplate
10	11	12	13	10	7	9	13	8	7	0.867692	98/100	NonOverlappingTemplate
14	11	13	10	4	12	12	9	5	10	0.383827	98/100	NonOverlappingTemplate
11	11	9	9	11	5	10	10	10	14	0.867692	99/100	NonOverlappingTemplate
8	9	13	14	14	10	6	8	7	11	0.574903	100/100	NonOverlappingTemplate
8	10	11	9	11	7	14	11	7	12	0.867692	98/100	NonOverlappingTemplate
10	10	13	10	9	6	11	9	8	14	0.851383	99/100	NonOverlappingTemplate
11	12	8	7	11	11	11	11	10	8	0.978072	99/100	NonOverlappingTemplate
9	10	11	12	12	10	10	7	10	9	0.991468	99/100	NonOverlappingTemplate
13	5	7	14	12	6	9	12	10	12	0.455937	98/100	NonOverlappingTemplate
9	15	8	13	12	8	10	7	6	12	0.574903	99/100	NonOverlappingTemplate
11	7	3	11	8	11	9	15	15	10	0.236810	99/100	NonOverlappingTemplate
7	8	14	10	7	10	10	11	9	14	0.779188	100/100	NonOverlappingTemplate
8	10	7	18	14	6	11	12	5	9	0.122325	99/100	NonOverlappingTemplate
10	13	6	7	9	8	11	9	15	12	0.637119	99/100	NonOverlappingTemplate
8	15	10	11	12	9	17	9	6	3	0.090936	100/100	NonOverlappingTemplate
10	10	11	9	11	8	7	7	16	11	0.719747	100/100	NonOverlappingTemplate
9	8	12	12	10	14	6	12	8	9	0.798139	99/100	NonOverlappingTemplate
8	10	7	13	7	9	10	14	13	9	0.759756	99/100	NonOverlappingTemplate
16	7	8	7	8	10	15	8	11	10	0.419021	100/100	NonOverlappingTemplate

13	15	9	5	9	11	9	13	9	7	0.514124	98/100	NonOverlappingTemplate
9	9	5	12	12	9	16	11	9	8	0.554420	96/100	NonOverlappingTemplate
8	9	9	8	9	11	13	17	8	8	0.554420	98/100	NonOverlappingTemplate
7	9	9	6	13	17	9	8	15	7	0.191687	99/100	NonOverlappingTemplate
8	11	7	10	12	11	8	12	11	10	0.971699	98/100	NonOverlappingTemplate
7	12	10	13	11	7	10	9	11	10	0.946308	99/100	NonOverlappingTemplate
8	7	9	14	13	11	10	9	9	10	0.897763	99/100	NonOverlappingTemplate
10	14	5	5	13	14	6	11	9	13	0.224821	99/100	NonOverlappingTemplate
14	7	10	13	12	12	10	5	9	8	0.616305	98/100	NonOverlappingTemplate
12	13	4	10	8	7	7	11	13	15	0.304126	100/100	NonOverlappingTemplate
17	6	9	13	7	9	13	9	7	10	0.319084	98/100	NonOverlappingTemplate
6	11	17	8	10	9	11	10	10	8	0.574903	100/100	NonOverlappingTemplate
16	3	5	16	11	14	14	9	4	8	0.008879	99/100	NonOverlappingTemplate
7	11	10	4	11	10	15	14	11	7	0.366918	100/100	NonOverlappingTemplate
8	9	7	10	11	10	12	11	9	13	0.964295	98/100	NonOverlappingTemplate
10	14	10	5	10	10	12	11	13	5	0.534146	98/100	NonOverlappingTemplate
9	11	10	11	11	10	4	11	11	12	0.867692	99/100	NonOverlappingTemplate
16	10	10	14	6	8	9	11	10	6	0.437274	100/100	NonOverlappingTemplate
10	14	10	6	11	9	6	10	8	16	0.437274	99/100	NonOverlappingTemplate
10	12	7	15	7	12	12	5	11	9	0.514124	99/100	NonOverlappingTemplate
8	14	10	10	6	10	7	11	13	11	0.779188	99/100	NonOverlappingTemplate
11	7	10	17	9	7	10	6	12	11	0.437274	98/100	NonOverlappingTemplate
6	11	10	10	8	10	17	9	12	7	0.494392	99/100	NonOverlappingTemplate
12	14	14	6	8	15	7	6	11	7	0.236810	100/100	NonOverlappingTemplate
8	8	11	6	11	14	10	9	10	13	0.816537	99/100	NonOverlappingTemplate
9	13	6	14	9	17	5	7	13	7	0.108791	100/100	NonOverlappingTemplate
10	5	9	12	12	10	8	10	9	15	0.699313	100/100	NonOverlappingTemplate
6	12	8	11	8	11	16	9	15	4	0.171867	99/100	NonOverlappingTemplate
6	13	11	11	12	10	15	11	3	8	0.275709	100/100	NonOverlappingTemplate
11	16	13	6	6	8	10	5	10	15	0.153763	100/100	NonOverlappingTemplate
8	10	9	14	11	9	11	13	5	10	0.759756	99/100	NonOverlappingTemplate
13	7	8	18	6	13	10	11	8	6	0.153763	99/100	NonOverlappingTemplate
8	5	15	12	13	9	10	9	12	7	0.514124	99/100	NonOverlappingTemplate
12	13	5	11	7	6	9	11	19	7	0.075719	96/100	NonOverlappingTemplate
9	10	10	15	11	11	10	12	4	8	0.616305	100/100	NonOverlappingTemplate
7	15	17	8	9	8	9	8	10	9	0.366918	99/100	NonOverlappingTemplate
11	3	14	11	6	6	9	16	8	16	0.040108	98/100	NonOverlappingTemplate
9	13	8	11	10	10	5	4	17	13	0.145326	99/100	NonOverlappingTemplate
7	15	13	8	10	11	13	4	9	10	0.401199	100/100	NonOverlappingTemplate
10	9	8	11	11	10	11	10	9	11	0.999438	100/100	NonOverlappingTemplate
6	13	6	16	10	12	11	11	8	7	0.383827	100/100	NonOverlappingTemplate
9	16	19	8	10	5	9	9	9	6	0.055361	99/100	NonOverlappingTemplate
10	6	12	7	10	11	11	10	9	14	0.851383	100/100	NonOverlappingTemplate
14	13	4	9	11	7	6	6	16	14	0.075719	99/100	NonOverlappingTemplate
9	14	6	8	11	9	16	6	12	9	0.383827	100/100	NonOverlappingTemplate
7	10	17	12	9	11	9	9	6	10	0.514124	99/100	NonOverlappingTemplate
9	11	12	6	10	13	11	10	6	12	0.816537	100/100	NonOverlappingTemplate
9	17	11	4	10	4	9	11	11	14	0.115387	98/100	NonOverlappingTemplate
13	8	10	7	6	13	10	13	9	11	0.759756	98/100	NonOverlappingTemplate
9	16	8	11	11	4	9	6	18	8	0.058984	97/100	NonOverlappingTemplate

6	10	12	13	13	7	4	17	7	11	0.115387	100/100	NonOverlappingTemplate
10	12	9	10	11	7	8	11	13	9	0.964295	98/100	NonOverlappingTemplate
11	11	6	11	10	10	7	9	13	12	0.897763	97/100	NonOverlappingTemplate
8	10	10	7	8	10	14	12	12	9	0.897763	100/100	NonOverlappingTemplate
11	14	13	7	8	7	12	14	6	8	0.455937	100/100	NonOverlappingTemplate
10	13	7	12	6	13	9	11	10	9	0.834308	98/100	NonOverlappingTemplate
3	18	12	7	14	6	9	8	10	13	0.045675	100/100	NonOverlappingTemplate
8	11	9	13	4	18	7	9	8	13	0.129620	99/100	NonOverlappingTemplate
8	10	9	14	9	9	12	12	5	12	0.739918	100/100	NonOverlappingTemplate
9	7	8	9	14	11	10	13	9	10	0.897763	98/100	NonOverlappingTemplate
8	12	8	19	9	6	9	8	13	8	0.171867	100/100	NonOverlappingTemplate
4	8	12	14	14	9	12	11	8	8	0.437274	100/100	NonOverlappingTemplate
10	7	9	8	7	9	10	16	10	14	0.574903	98/100	NonOverlappingTemplate
11	15	11	8	8	11	9	9	5	13	0.616305	98/100	NonOverlappingTemplate
7	6	10	10	12	9	19	9	5	13	0.102526	100/100	NonOverlappingTemplate
11	7	3	11	8	11	9	15	15	10	0.236810	99/100	NonOverlappingTemplate
12	18	7	10	5	15	11	8	9	5	0.071177	99/100	OverlappingTemplate
8	5	11	10	5	10	13	14	19	5	0.028817	99/100	Universal
7	10	12	13	8	11	7	9	12	11	0.897763	99/100	ApproximateEntropy
5	6	7	4	5	8	9	8	5	7	0.888137	64/64	RandomExcursions
3	7	8	6	9	7	4	5	9	6	0.706149	64/64	RandomExcursions
6	8	7	7	7	4	9	8	3	5	0.772760	63/64	RandomExcursions
6	11	5	5	4	10	7	8	4	4	0.350485	63/64	RandomExcursions
6	6	5	8	11	5	8	5	7	3	0.568055	64/64	RandomExcursions
4	9	7	4	6	6	6	7	12	3	0.299251	64/64	RandomExcursions
5	9	6	8	7	4	4	8	6	7	0.862344	64/64	RandomExcursions
2	5	9	7	6	7	7	5	11	5	0.407091	63/64	RandomExcursions
3	10	3	6	4	6	4	11	7	10	0.122325	64/64	RandomExcursionsVariant
4	4	8	7	5	4	4	10	11	7	0.299251	64/64	RandomExcursionsVariant
5	2	12	7	4	2	6	9	7	10	0.054199	64/64	RandomExcursionsVariant
6	6	4	5	9	2	9	6	9	8	0.468595	63/64	RandomExcursionsVariant
7	7	3	6	5	10	6	9	7	4	0.637119	63/64	RandomExcursionsVariant
5	8	5	5	6	10	8	7	7	3	0.706149	63/64	RandomExcursionsVariant
5	4	8	4	4	5	8	7	10	9	0.534146	63/64	RandomExcursionsVariant
5	6	5	9	3	4	8	11	6	7	0.437274	64/64	RandomExcursionsVariant
4	7	6	9	8	6	5	11	3	5	0.437274	64/64	RandomExcursionsVariant
4	7	12	4	7	10	8	6	3	3	0.122325	64/64	RandomExcursionsVariant
11	7	6	8	6	2	4	8	6	6	0.437274	64/64	RandomExcursionsVariant
9	9	7	3	4	8	7	6	2	9	0.324180	62/64	RandomExcursionsVariant
9	7	7	5	6	6	2	6	8	8	0.739918	62/64	RandomExcursionsVariant
9	8	7	1	6	9	5	9	5	5	0.350485	62/64	RandomExcursionsVariant
12	7	8	3	6	5	7	7	5	4	0.378138	62/64	RandomExcursionsVariant
13	5	8	7	1	7	5	11	3	4	0.017912	63/64	RandomExcursionsVariant
11	5	8	4	7	10	5	5	7	2	0.232760	64/64	RandomExcursionsVariant
9	10	3	9	7	4	6	7	1	8	0.162606	63/64	RandomExcursionsVariant
6	10	8	11	6	14	9	10	13	13	0.616305	99/100	Serial
14	9	9	7	5	10	10	12	12	12	0.699313	99/100	Serial
10	10	11	11	11	10	9	9	12	7	0.994250	100/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 60 for a sample size = 64 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.6: Risultati del test NIST STS 2.1 su una sequenza di $100 \times 10^6\,\mathrm{bit}$ prelevati in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso E.

A.10 Risultati test statistici NIST - Caso G

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <prandom.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  0   0   0   0   0   0   0   0   0 100  0.000000 *  100/100     Frequency
  0   0   0   0   0   0   0   0   0 100  0.000000 *  100/100     BlockFrequency
  0   0   0   0   0   0   0   0   0 100  0.000000 *  100/100     CumulativeSums
  0   0   0   0   0   0   0   0   0 100  0.000000 *  100/100     CumulativeSums
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Runs
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  LongestRun
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Rank
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  FFT
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
```

```
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
100  0  0  0  0  0  0  0  0  0  0.000000 *    0/100  *  NonOverlappingTemplate
```

```
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
```

```
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  NonOverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  OverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Universal
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  ApproximateEntropy
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursions
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0 100  0.000000 *  100/100     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   0     ----     ------     RandomExcursionsVariant
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Serial
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Serial
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  LinearComplexity
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test is undefined.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.7: Risultati del test NIST STS 2.1 su una sequenza di 100×10^6 bit. La sequenza analizzata, prelevata in ingresso all'entropy distiller di Linux™, è deterministica. Essa viene generata mediante uno script, replicando iterativamente il pattern di bit `01`.

A.11 Risultati test statistici NIST - Caso H

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <prandom_pool.txt>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 13   9   9  11  10   8  13   9  11   7  0.935716  100/100      Frequency
 10  16  12   6  13   9  13   9   6   6  0.289667  100/100      BlockFrequency
  9  11  11   6  12  12  11   9  10   9  0.964295  100/100      CumulativeSums
 12  13   5  11  13   9   7   7  11  12  0.616305   99/100      CumulativeSums
  7   9  10  13  12  12  10  11   7   9  0.924076   98/100      Runs
  9  10   9  13   8  14  10  13   7   7  0.759756   99/100      LongestRun
  9  10  10   7   9  11  14  17   5   8  0.304126   99/100      Rank
 13  11  10  12   6   8  12  10   9   9  0.911413   98/100      FFT
  8  13  14  10  11   6  10  10  10   8  0.834308  100/100      NonOverlappingTemplate
  9   8   8  14  10  13  12   8   7  11  0.816537  100/100      NonOverlappingTemplate
 15   8  10   7  11   9  16   5   9  10  0.334538   98/100      NonOverlappingTemplate
 15   9  18   7  10  12   7  11   7   4  0.071177  100/100      NonOverlappingTemplate
  9  11   8   9  16   8  14   6  13   6  0.319084   98/100      NonOverlappingTemplate
 16   6   7  10  12  13   8   9   6  13  0.319084   98/100      NonOverlappingTemplate
 11  14  11  10   4   3  10  15   9  13  0.129620   99/100      NonOverlappingTemplate
 12  10  15   9  11  14   6   5   9   9  0.437274   95/100  *   NonOverlappingTemplate
 11  11  10   7  10  16   2   6  18   9  0.023545  100/100      NonOverlappingTemplate
 11  12  15   8  11   5   7  14   8   9  0.437274  100/100      NonOverlappingTemplate
  8  13  12  14  10  12   6  11   5   9  0.534146   99/100      NonOverlappingTemplate
  8  13  12  10   5  13   8  13  11   7  0.595549   99/100      NonOverlappingTemplate
  9  10   9   8   8  16  11  11  12   6  0.657933  100/100      NonOverlappingTemplate
  7  11   6  11  16   4  13  12  12   8  0.213309   99/100      NonOverlappingTemplate
 16   8  10  11  11   9   4  12  10   9  0.494392  100/100      NonOverlappingTemplate
 13   8  13   8  13  12   4  10   8  11  0.534146   99/100      NonOverlappingTemplate
 13  11   6   7   9  11  13  11  12   7  0.739918   96/100      NonOverlappingTemplate
  6   8  14  13  10  11   7  14   6  11  0.455937  100/100      NonOverlappingTemplate
 10   8  10  13  10  12  15   7   6   9  0.657933   99/100      NonOverlappingTemplate
  9  11   9   6  11   7   7  13  14  13  0.616305  100/100      NonOverlappingTemplate
  6  19  13   8   8  13   6   7   6  14  0.035174  100/100      NonOverlappingTemplate
 10  12  10  11   9   5  11  15  11   6  0.595549  100/100      NonOverlappingTemplate
  5  12   9  12  11  11   9  14  10   7  0.719747   99/100      NonOverlappingTemplate
  7  13   7  10  14   8   6   8  11  16  0.319084   97/100      NonOverlappingTemplate
  9  11  13   7   8  10  12  13  10   7  0.867692  100/100      NonOverlappingTemplate
  1  14  14   8  11  11  12  10   9  10  0.191687  100/100      NonOverlappingTemplate
 12  13   9   8   9  10   7  11  11  10  0.964295   99/100      NonOverlappingTemplate
  9   8   8   9   9   9  11  10  17  10  0.719747  100/100      NonOverlappingTemplate
 11   7  13  11  10   9  13   7   8  11  0.883171   98/100      NonOverlappingTemplate
  7  10   5  10  13  11  11  11   6  16  0.366918   99/100      NonOverlappingTemplate
 13  10  10   7  13  13  10   8  10   6  0.779188   99/100      NonOverlappingTemplate
 11   8  11  11  11  12   6  12  10   8  0.935716   99/100      NonOverlappingTemplate
```

8	7	12	14	8	6	14	9	14	8	0.437274	100/100		NonOverlappingTemplate
11	8	5	17	14	13	5	7	10	10	0.129620	99/100		NonOverlappingTemplate
12	8	9	11	8	13	9	3	12	15	0.334538	99/100		NonOverlappingTemplate
8	11	12	11	10	12	12	9	9	6	0.935716	100/100		NonOverlappingTemplate
18	6	9	9	8	9	11	10	11	9	0.437274	95/100	*	NonOverlappingTemplate
10	10	10	5	10	9	9	15	15	7	0.474986	98/100		NonOverlappingTemplate
8	13	9	10	7	7	14	14	9	9	0.678686	99/100		NonOverlappingTemplate
5	8	9	12	10	15	10	12	10	9	0.699313	100/100		NonOverlappingTemplate
6	15	9	9	13	9	8	11	11	9	0.739918	100/100		NonOverlappingTemplate
4	12	15	13	8	9	15	8	11	5	0.145326	100/100		NonOverlappingTemplate
9	11	7	10	10	8	14	11	11	9	0.946308	100/100		NonOverlappingTemplate
10	7	10	11	12	7	8	9	17	9	0.554420	99/100		NonOverlappingTemplate
11	10	8	14	10	7	8	16	7	9	0.534146	100/100		NonOverlappingTemplate
10	8	9	7	15	8	14	8	10	11	0.699313	99/100		NonOverlappingTemplate
10	5	11	16	14	12	6	9	10	7	0.289667	99/100		NonOverlappingTemplate
12	14	10	9	8	5	10	17	6	9	0.236810	98/100		NonOverlappingTemplate
6	7	13	11	13	13	5	11	7	14	0.319084	100/100		NonOverlappingTemplate
9	10	13	14	13	8	7	12	9	5	0.554420	100/100		NonOverlappingTemplate
11	12	9	13	11	8	10	8	10	8	0.971699	98/100		NonOverlappingTemplate
8	9	13	10	12	11	10	10	12	5	0.851383	98/100		NonOverlappingTemplate
16	11	9	10	10	13	10	4	6	11	0.350485	100/100		NonOverlappingTemplate
9	18	10	10	10	13	4	11	8	7	0.191687	100/100		NonOverlappingTemplate
13	12	10	8	5	12	12	10	11	7	0.739918	98/100		NonOverlappingTemplate
11	10	6	13	13	10	8	10	8	11	0.883171	100/100		NonOverlappingTemplate
10	12	9	6	8	11	12	11	8	13	0.883171	99/100		NonOverlappingTemplate
6	8	16	11	8	7	15	8	12	9	0.319084	98/100		NonOverlappingTemplate
16	10	9	7	4	10	11	10	9	14	0.350485	98/100		NonOverlappingTemplate
10	16	13	10	7	4	5	16	9	10	0.085587	100/100		NonOverlappingTemplate
14	11	6	12	6	9	12	9	9	12	0.699313	99/100		NonOverlappingTemplate
11	10	7	8	11	9	12	8	15	9	0.834308	100/100		NonOverlappingTemplate
13	8	8	9	12	13	5	13	11	8	0.637119	99/100		NonOverlappingTemplate
8	8	11	10	11	11	10	12	9	10	0.996335	100/100		NonOverlappingTemplate
9	9	16	7	7	8	12	12	11	9	0.637119	98/100		NonOverlappingTemplate
13	4	10	9	12	8	12	9	9	14	0.574903	99/100		NonOverlappingTemplate
7	12	11	5	5	14	9	9	18	10	0.102526	99/100		NonOverlappingTemplate
13	9	13	9	10	9	10	6	11	10	0.924076	99/100		NonOverlappingTemplate
9	15	8	11	9	11	9	16	8	4	0.275709	99/100		NonOverlappingTemplate
15	10	12	5	11	8	9	7	14	9	0.474986	99/100		NonOverlappingTemplate
9	8	14	8	6	11	4	10	15	15	0.171867	99/100		NonOverlappingTemplate
11	12	10	13	9	13	13	3	8	8	0.437274	100/100		NonOverlappingTemplate
6	6	8	8	14	6	18	14	7	13	0.048716	100/100		NonOverlappingTemplate
11	9	11	9	13	10	5	8	12	12	0.834308	98/100		NonOverlappingTemplate
8	13	14	10	11	6	10	10	10	8	0.834308	100/100		NonOverlappingTemplate
6	9	12	8	10	10	12	13	8	12	0.867692	100/100		NonOverlappingTemplate
13	6	12	11	11	9	8	12	9	9	0.897763	98/100		NonOverlappingTemplate
7	9	11	14	11	7	10	11	12	8	0.867692	98/100		NonOverlappingTemplate
15	12	13	13	9	9	5	6	10	8	0.401199	100/100		NonOverlappingTemplate
17	9	11	12	7	8	10	6	11	9	0.474986	98/100		NonOverlappingTemplate
8	11	10	14	10	9	8	9	9	12	0.955835	100/100		NonOverlappingTemplate
11	11	12	9	7	8	12	12	12	6	0.851383	99/100		NonOverlappingTemplate

7	14	11	11	8	11	13	11	8	6	0.719747	100/100	NonOverlappingTemplate
10	11	12	9	8	10	9	5	13	13	0.798139	99/100	NonOverlappingTemplate
12	9	7	9	10	11	7	11	10	14	0.897763	98/100	NonOverlappingTemplate
8	15	5	7	12	7	14	13	9	10	0.334538	100/100	NonOverlappingTemplate
9	13	11	10	9	11	9	11	6	11	0.955835	100/100	NonOverlappingTemplate
6	13	13	7	9	12	6	13	13	8	0.474986	100/100	NonOverlappingTemplate
20	5	5	12	11	6	11	11	8	11	0.037566	97/100	NonOverlappingTemplate
8	10	14	18	8	7	12	9	6	8	0.202268	99/100	NonOverlappingTemplate
9	12	5	8	9	11	10	10	14	12	0.779188	100/100	NonOverlappingTemplate
7	9	11	12	13	7	9	12	7	13	0.779188	100/100	NonOverlappingTemplate
9	9	6	10	12	16	15	7	6	10	0.289667	99/100	NonOverlappingTemplate
11	10	6	5	7	12	11	14	9	15	0.366918	98/100	NonOverlappingTemplate
11	5	10	5	11	14	8	10	7	19	0.062821	97/100	NonOverlappingTemplate
7	7	14	9	11	9	13	9	9	12	0.816537	99/100	NonOverlappingTemplate
12	8	9	8	14	13	9	5	11	11	0.678686	99/100	NonOverlappingTemplate
15	7	6	12	7	14	10	12	14	3	0.096578	98/100	NonOverlappingTemplate
11	8	14	10	10	9	12	6	13	7	0.739918	100/100	NonOverlappingTemplate
10	9	12	12	8	16	7	8	8	10	0.678686	99/100	NonOverlappingTemplate
9	12	8	6	9	9	11	10	16	10	0.699313	100/100	NonOverlappingTemplate
6	10	13	11	8	9	10	13	12	8	0.851383	100/100	NonOverlappingTemplate
14	5	11	9	11	15	7	9	14	5	0.213309	96/100	NonOverlappingTemplate
4	13	12	12	5	10	13	8	11	12	0.383827	99/100	NonOverlappingTemplate
13	11	8	11	5	11	10	10	8	13	0.798139	99/100	NonOverlappingTemplate
15	7	10	7	10	9	5	14	13	10	0.401199	96/100	NonOverlappingTemplate
9	10	10	7	13	11	7	13	10	10	0.924076	99/100	NonOverlappingTemplate
8	15	7	9	10	12	8	10	8	13	0.739918	98/100	NonOverlappingTemplate
6	4	12	9	5	18	9	15	13	9	0.032923	99/100	NonOverlappingTemplate
6	5	6	13	12	8	13	11	14	12	0.319084	99/100	NonOverlappingTemplate
11	11	15	12	5	10	6	12	12	6	0.383827	99/100	NonOverlappingTemplate
15	9	12	8	12	10	4	10	6	14	0.304126	100/100	NonOverlappingTemplate
12	8	16	11	10	8	10	9	10	6	0.678686	100/100	NonOverlappingTemplate
17	12	14	7	6	9	13	13	4	5	0.042808	97/100	NonOverlappingTemplate
16	10	7	11	9	6	10	12	11	8	0.616305	99/100	NonOverlappingTemplate
6	11	11	10	13	8	7	12	10	12	0.851383	100/100	NonOverlappingTemplate
8	9	10	7	12	12	6	14	11	11	0.779188	100/100	NonOverlappingTemplate
11	10	10	8	8	10	8	15	11	9	0.911413	99/100	NonOverlappingTemplate
12	11	7	13	9	10	13	7	12	6	0.719747	99/100	NonOverlappingTemplate
9	10	11	12	9	15	8	15	6	5	0.334538	100/100	NonOverlappingTemplate
6	12	16	11	5	11	6	14	11	8	0.213309	100/100	NonOverlappingTemplate
10	8	7	5	14	13	13	8	16	6	0.171867	100/100	NonOverlappingTemplate
10	11	9	10	11	8	8	19	6	8	0.262249	99/100	NonOverlappingTemplate
9	9	14	9	13	6	16	5	9	10	0.304126	100/100	NonOverlappingTemplate
6	9	8	14	9	11	6	10	14	13	0.534146	100/100	NonOverlappingTemplate
13	8	6	12	7	9	13	8	9	15	0.514124	100/100	NonOverlappingTemplate
11	6	9	11	16	9	12	6	10	10	0.574903	99/100	NonOverlappingTemplate
8	11	11	9	8	11	7	11	4	20	0.071177	100/100	NonOverlappingTemplate
12	5	6	13	10	13	4	18	9	10	0.058984	99/100	NonOverlappingTemplate
8	11	11	11	9	10	12	11	11	6	0.964295	99/100	NonOverlappingTemplate
10	10	9	15	8	9	12	10	5	12	0.699313	100/100	NonOverlappingTemplate
7	10	6	7	13	13	12	10	15	7	0.437274	100/100	NonOverlappingTemplate

10	9	13	14	7	8	11	11	10	7	0.834308	99/100	NonOverlappingTemplate
7	7	9	14	7	13	11	11	11	10	0.779188	100/100	NonOverlappingTemplate
8	11	10	8	10	7	10	10	13	13	0.935716	99/100	NonOverlappingTemplate
7	10	10	6	8	7	13	13	9	17	0.304126	98/100	NonOverlappingTemplate
11	8	8	8	12	15	11	10	7	10	0.816537	97/100	NonOverlappingTemplate
5	10	9	9	16	8	11	15	9	8	0.366918	99/100	NonOverlappingTemplate
17	7	7	13	9	9	11	15	7	5	0.129620	97/100	NonOverlappingTemplate
13	10	9	12	9	7	6	10	12	12	0.851383	99/100	NonOverlappingTemplate
13	9	13	12	6	9	11	11	8	8	0.834308	99/100	NonOverlappingTemplate
7	7	13	10	7	15	9	10	12	10	0.678686	100/100	NonOverlappingTemplate
13	8	11	10	9	12	10	9	11	7	0.964295	98/100	NonOverlappingTemplate
8	9	12	14	12	10	8	10	12	5	0.719747	98/100	NonOverlappingTemplate
5	12	13	6	15	7	13	10	9	10	0.366918	99/100	NonOverlappingTemplate
7	11	14	5	12	13	10	10	10	8	0.657933	99/100	NonOverlappingTemplate
10	11	8	10	11	7	16	7	12	8	0.657933	100/100	NonOverlappingTemplate
11	9	11	9	13	10	5	8	12	12	0.834308	98/100	NonOverlappingTemplate
14	12	7	8	11	17	9	7	6	9	0.275709	97/100	OverlappingTemplate
4	14	10	10	4	7	18	15	10	8	0.025193	99/100	Universal
11	9	9	8	12	5	16	10	12	8	0.534146	99/100	ApproximateEntropy
7	2	4	7	6	5	12	5	8	5	0.287306	60/61	RandomExcursions
5	8	8	8	10	5	5	4	2	6	0.484646	60/61	RandomExcursions
7	10	3	8	4	7	2	3	7	10	0.170294	60/61	RandomExcursions
8	10	3	3	5	5	7	6	11	3	0.186566	61/61	RandomExcursions
1	9	6	6	3	8	6	3	8	11	0.116519	61/61	RandomExcursions
7	7	1	8	4	6	9	8	4	7	0.452799	61/61	RandomExcursions
5	3	2	7	11	6	3	10	10	4	0.063482	61/61	RandomExcursions
4	4	9	4	4	10	6	9	6	5	0.484646	61/61	RandomExcursions
4	6	8	5	6	8	4	7	7	6	0.957319	61/61	RandomExcursionsVariant
5	7	3	5	8	7	7	4	6	9	0.819544	61/61	RandomExcursionsVariant
5	5	4	2	12	4	8	7	5	9	0.170294	61/61	RandomExcursionsVariant
8	2	3	7	8	9	5	5	7	7	0.551026	60/61	RandomExcursionsVariant
9	2	5	6	6	11	4	7	8	3	0.242986	60/61	RandomExcursionsVariant
8	5	4	7	5	8	6	6	6	6	0.980883	60/61	RandomExcursionsVariant
6	6	3	6	10	3	11	5	6	5	0.337162	60/61	RandomExcursionsVariant
7	7	7	2	8	8	4	8	2	8	0.422034	60/61	RandomExcursionsVariant
9	4	9	4	4	8	5	7	5	6	0.723129	61/61	RandomExcursionsVariant
4	7	4	9	4	5	8	7	7	6	0.848588	61/61	RandomExcursionsVariant
7	3	7	7	1	11	12	5	3	5	0.033288	61/61	RandomExcursionsVariant
9	5	3	10	2	6	5	7	7	7	0.422034	61/61	RandomExcursionsVariant
8	6	4	3	3	6	6	9	8	8	0.619772	60/61	RandomExcursionsVariant
6	7	3	5	5	7	6	6	10	6	0.848588	59/61	RandomExcursionsVariant
7	4	3	8	7	6	6	6	4	10	0.689019	60/61	RandomExcursionsVariant
4	5	6	10	8	4	4	8	5	7	0.689019	61/61	RandomExcursionsVariant
4	7	4	9	7	7	6	4	7	6	0.900104	61/61	RandomExcursionsVariant
5	5	9	5	6	8	5	4	5	9	0.819544	61/61	RandomExcursionsVariant
7	13	13	10	7	10	10	2	14	14	0.153763	100/100	Serial
8	6	11	11	13	6	12	9	10	14	0.657933	99/100	Serial
10	12	9	9	7	14	10	9	11	9	0.946308	99/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 58 for a sample size = 61 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.8: Risultati del test NIST STS 2.1 su una sequenza di 100×10^6 bit prelevati in uscita dall'entropy distiller di Linux™, nelle stesse condizioni del caso G.

A.12 Risultati test statistici NIST - Caso XOR

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <device_raw_N3K4F1_alt_code_unbias1.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
16	6	7	6	9	9	11	9	8	19	0.055361	98/100	Frequency
12	10	5	15	14	10	6	13	8	7	0.289667	99/100	BlockFrequency
20	8	4	6	4	15	7	15	7	14	0.001112	98/100	CumulativeSums
16	5	7	13	4	12	8	8	16	11	0.058984	99/100	CumulativeSums
13	13	10	13	10	9	10	8	6	8	0.816537	100/100	Runs
14	13	5	8	13	11	8	10	7	11	0.554420	97/100	LongestRun
13	7	13	11	11	4	7	17	12	5	0.085587	98/100	Rank
12	18	8	8	6	7	8	10	12	11	0.275709	96/100	FFT
13	10	8	11	11	6	10	14	6	11	0.699313	98/100	NonOverlappingTemplate
22	8	8	5	7	9	15	8	6	12	0.004981	97/100	NonOverlappingTemplate
10	13	13	10	11	11	10	10	5	7	0.798139	99/100	NonOverlappingTemplate
6	9	13	11	13	10	12	7	10	9	0.834308	100/100	NonOverlappingTemplate
12	11	6	16	9	11	11	11	7	6	0.474986	98/100	NonOverlappingTemplate
11	10	11	15	11	10	4	10	9	9	0.678686	100/100	NonOverlappingTemplate
6	9	13	14	16	10	6	10	9	7	0.319084	99/100	NonOverlappingTemplate
13	9	10	5	11	11	11	14	10	6	0.637119	98/100	NonOverlappingTemplate
4	10	12	10	7	8	9	11	16	13	0.350485	99/100	NonOverlappingTemplate
10	18	4	12	10	11	8	10	6	11	0.181557	100/100	NonOverlappingTemplate
12	9	5	11	10	7	7	17	14	8	0.224821	98/100	NonOverlappingTemplate
6	5	8	14	14	11	11	14	8	9	0.350485	100/100	NonOverlappingTemplate
11	13	13	9	17	10	8	5	6	8	0.224821	100/100	NonOverlappingTemplate
6	6	7	10	8	13	12	14	12	12	0.514124	99/100	NonOverlappingTemplate
10	16	4	9	6	12	9	18	7	9	0.051942	99/100	NonOverlappingTemplate
14	8	9	11	7	9	11	15	9	7	0.657933	98/100	NonOverlappingTemplate
8	12	10	10	8	12	6	8	13	13	0.798139	100/100	NonOverlappingTemplate
11	10	8	13	7	8	12	8	13	10	0.883171	99/100	NonOverlappingTemplate
13	9	7	7	13	13	7	13	8	10	0.657933	99/100	NonOverlappingTemplate
6	10	11	9	7	9	9	14	10	15	0.637119	100/100	NonOverlappingTemplate
7	12	11	12	8	6	13	10	9	12	0.816537	100/100	NonOverlappingTemplate
8	6	5	15	14	6	7	12	13	14	0.122325	98/100	NonOverlappingTemplate
4	8	10	12	8	11	8	14	15	10	0.401199	100/100	NonOverlappingTemplate
7	11	11	6	8	8	16	15	8	10	0.350485	100/100	NonOverlappingTemplate
10	9	10	8	10	8	15	6	12	12	0.759756	99/100	NonOverlappingTemplate
13	5	9	13	14	5	7	12	13	9	0.289667	100/100	NonOverlappingTemplate
10	13	3	13	9	17	6	8	11	10	0.129620	98/100	NonOverlappingTemplate
10	8	7	5	9	12	14	10	10	15	0.494392	100/100	NonOverlappingTemplate
8	15	0	13	6	9	9	17	10	13	0.010988	100/100	NonOverlappingTemplate
7	7	9	14	14	15	8	11	6	9	0.366918	100/100	NonOverlappingTemplate
10	14	12	12	8	8	8	9	6	13	0.719747	100/100	NonOverlappingTemplate
8	17	4	8	8	14	12	9	9	11	0.213309	100/100	NonOverlappingTemplate

13	6	7	8	16	14	5	5	14	12	0.066882	100/100		NonOverlappingTemplate
10	9	12	12	15	8	9	9	12	4	0.534146	98/100		NonOverlappingTemplate
14	11	9	9	7	14	11	7	6	12	0.595549	98/100		NonOverlappingTemplate
11	11	8	7	9	15	9	9	9	12	0.851383	98/100		NonOverlappingTemplate
12	7	9	11	7	9	14	11	15	5	0.419021	98/100		NonOverlappingTemplate
12	12	7	13	15	11	7	5	12	6	0.304126	100/100		NonOverlappingTemplate
9	12	14	8	7	11	8	12	7	12	0.779188	98/100		NonOverlappingTemplate
6	8	6	16	8	8	17	16	5	10	0.025193	99/100		NonOverlappingTemplate
5	15	6	11	11	13	14	6	8	11	0.249284	100/100		NonOverlappingTemplate
9	10	11	4	11	13	13	11	9	9	0.739918	99/100		NonOverlappingTemplate
14	10	10	8	7	8	9	11	10	13	0.883171	100/100		NonOverlappingTemplate
11	7	10	9	6	9	8	14	11	15	0.595549	99/100		NonOverlappingTemplate
8	10	7	4	9	14	13	14	10	11	0.419021	100/100		NonOverlappingTemplate
7	8	9	8	12	11	9	12	11	13	0.924076	98/100		NonOverlappingTemplate
9	9	10	9	9	8	13	7	13	13	0.883171	99/100		NonOverlappingTemplate
17	7	7	10	10	9	16	7	8	9	0.224821	97/100		NonOverlappingTemplate
7	5	9	22	8	8	6	15	11	9	0.006196	99/100		NonOverlappingTemplate
5	11	9	17	14	11	13	6	6	8	0.129620	100/100		NonOverlappingTemplate
10	9	9	9	11	10	9	9	12	12	0.997823	97/100		NonOverlappingTemplate
10	11	12	11	11	8	10	9	9	9	0.997823	99/100		NonOverlappingTemplate
12	11	12	9	11	15	8	11	7	4	0.474986	97/100		NonOverlappingTemplate
11	12	13	10	10	6	7	12	8	11	0.851383	100/100		NonOverlappingTemplate
10	12	8	6	9	13	11	9	12	10	0.911413	98/100		NonOverlappingTemplate
13	15	12	7	8	7	7	7	13	11	0.455937	99/100		NonOverlappingTemplate
11	11	13	8	9	12	9	8	11	8	0.964295	100/100		NonOverlappingTemplate
7	11	9	4	12	19	16	6	7	9	0.021999	99/100		NonOverlappingTemplate
10	11	6	5	10	12	9	8	11	18	0.236810	96/100		NonOverlappingTemplate
20	9	14	10	2	11	9	5	8	12	0.010237	95/100	*	NonOverlappingTemplate
8	7	10	10	15	11	10	11	2	16	0.122325	98/100		NonOverlappingTemplate
6	17	6	4	11	10	12	13	9	12	0.137282	99/100		NonOverlappingTemplate
8	9	14	11	12	7	11	11	12	5	0.678686	99/100		NonOverlappingTemplate
10	14	7	6	12	6	16	9	10	10	0.366918	100/100		NonOverlappingTemplate
10	12	13	11	10	7	6	9	10	12	0.883171	99/100		NonOverlappingTemplate
8	12	7	6	16	8	7	12	11	13	0.383827	99/100		NonOverlappingTemplate
9	9	12	5	9	13	9	12	11	11	0.851383	99/100		NonOverlappingTemplate
13	8	6	12	8	7	15	12	13	6	0.350485	98/100		NonOverlappingTemplate
12	14	12	8	7	7	9	8	9	14	0.657933	100/100		NonOverlappingTemplate
9	1	14	11	14	5	8	7	18	13	0.007160	100/100		NonOverlappingTemplate
13	11	4	13	8	12	11	11	9	8	0.637119	98/100		NonOverlappingTemplate
13	14	11	8	12	8	9	8	9	8	0.851383	98/100		NonOverlappingTemplate
10	5	10	11	12	9	9	9	15	10	0.759756	99/100		NonOverlappingTemplate
9	9	7	7	15	10	13	11	7	12	0.657933	100/100		NonOverlappingTemplate
13	10	8	11	11	6	10	14	6	11	0.699313	98/100		NonOverlappingTemplate
8	9	11	20	15	9	9	4	7	8	0.032923	100/100		NonOverlappingTemplate
12	11	17	5	11	8	7	9	12	8	0.334538	96/100		NonOverlappingTemplate
11	12	6	12	12	11	7	9	11	9	0.897763	100/100		NonOverlappingTemplate
12	15	9	9	9	11	7	10	11	7	0.816537	99/100		NonOverlappingTemplate
5	9	11	6	4	17	9	16	10	13	0.042808	100/100		NonOverlappingTemplate
15	8	9	9	10	8	9	12	13	7	0.759756	99/100		NonOverlappingTemplate
9	8	10	10	12	16	11	9	5	10	0.616305	98/100		NonOverlappingTemplate

9	8	7	10	10	10	12	12	13	9	0.955835	100/100	NonOverlappingTemplate
11	8	9	14	10	12	9	7	12	8	0.883171	99/100	NonOverlappingTemplate
9	8	11	15	10	7	12	10	14	4	0.383827	100/100	NonOverlappingTemplate
18	10	9	11	11	8	10	5	9	9	0.366918	98/100	NonOverlappingTemplate
5	14	12	15	11	8	8	12	7	8	0.383827	100/100	NonOverlappingTemplate
15	8	11	12	8	12	10	7	11	6	0.657933	99/100	NonOverlappingTemplate
9	9	8	14	13	5	11	17	9	5	0.153763	100/100	NonOverlappingTemplate
7	8	6	8	12	12	15	13	8	11	0.534146	99/100	NonOverlappingTemplate
11	13	7	13	9	11	7	9	13	7	0.759756	99/100	NonOverlappingTemplate
17	7	4	6	15	9	10	13	11	8	0.090936	99/100	NonOverlappingTemplate
11	5	10	12	13	6	11	12	12	8	0.657933	99/100	NonOverlappingTemplate
9	9	11	14	13	11	8	5	9	11	0.739918	99/100	NonOverlappingTemplate
12	10	13	13	9	10	3	11	9	10	0.595549	100/100	NonOverlappingTemplate
16	11	6	11	14	12	9	7	7	7	0.334538	98/100	NonOverlappingTemplate
11	4	11	11	10	16	10	10	13	4	0.213309	99/100	NonOverlappingTemplate
12	13	5	5	16	10	10	10	7	12	0.262249	98/100	NonOverlappingTemplate
10	8	9	12	8	13	15	7	6	12	0.574903	98/100	NonOverlappingTemplate
10	14	13	7	10	10	7	8	13	8	0.739918	99/100	NonOverlappingTemplate
7	10	9	14	14	5	11	12	7	11	0.514124	99/100	NonOverlappingTemplate
10	14	8	12	10	11	8	11	7	9	0.911413	98/100	NonOverlappingTemplate
6	15	9	10	7	9	16	8	8	12	0.350485	100/100	NonOverlappingTemplate
12	7	8	14	9	10	11	8	8	13	0.816537	100/100	NonOverlappingTemplate
10	14	7	20	13	8	9	4	11	4	0.011791	100/100	NonOverlappingTemplate
9	10	13	4	11	8	12	7	11	15	0.437274	99/100	NonOverlappingTemplate
12	7	4	15	10	12	11	15	4	10	0.122325	99/100	NonOverlappingTemplate
11	10	10	14	10	11	9	4	9	12	0.739918	98/100	NonOverlappingTemplate
13	12	9	8	6	13	8	15	6	10	0.455937	100/100	NonOverlappingTemplate
5	9	8	18	8	9	13	8	9	13	0.202268	100/100	NonOverlappingTemplate
14	16	14	10	8	2	11	5	10	10	0.062821	97/100	NonOverlappingTemplate
10	14	9	8	13	9	13	9	7	8	0.798139	97/100	NonOverlappingTemplate
9	9	9	13	9	10	12	7	15	7	0.739918	100/100	NonOverlappingTemplate
10	9	10	9	12	13	12	5	12	8	0.816537	99/100	NonOverlappingTemplate
10	11	5	8	13	15	8	13	11	6	0.401199	100/100	NonOverlappingTemplate
8	9	4	9	12	15	16	7	14	6	0.096578	100/100	NonOverlappingTemplate
9	7	8	7	12	10	15	8	17	7	0.249284	100/100	NonOverlappingTemplate
12	12	10	10	8	11	8	11	8	10	0.987896	99/100	NonOverlappingTemplate
12	8	9	10	10	7	11	14	8	11	0.911413	100/100	NonOverlappingTemplate
10	7	7	11	9	11	8	13	11	13	0.883171	99/100	NonOverlappingTemplate
6	13	8	14	9	11	12	9	13	5	0.474986	100/100	NonOverlappingTemplate
8	9	11	10	10	12	8	10	13	9	0.983453	97/100	NonOverlappingTemplate
10	13	7	10	11	4	13	11	13	8	0.554420	99/100	NonOverlappingTemplate
10	16	8	7	9	9	16	8	12	5	0.213309	99/100	NonOverlappingTemplate
13	6	11	13	4	10	9	11	9	14	0.437274	98/100	NonOverlappingTemplate
6	15	8	9	11	16	5	8	8	14	0.153763	100/100	NonOverlappingTemplate
14	10	5	10	11	13	15	5	6	11	0.224821	99/100	NonOverlappingTemplate
11	9	11	10	8	7	18	5	13	8	0.224821	100/100	NonOverlappingTemplate
11	8	9	8	8	13	7	11	14	11	0.834308	98/100	NonOverlappingTemplate
15	8	7	10	10	10	10	10	11	9	0.911413	99/100	NonOverlappingTemplate
9	14	10	10	12	6	7	8	14	10	0.678686	97/100	NonOverlappingTemplate
7	12	7	15	6	9	12	14	11	7	0.401199	100/100	NonOverlappingTemplate

9	11	12	9	9	10	8	11	9	12	0.994250	99/100	NonOverlappingTemplate
13	7	12	4	16	12	14	8	8	6	0.129620	100/100	NonOverlappingTemplate
5	8	12	10	13	10	11	12	8	11	0.816537	99/100	NonOverlappingTemplate
10	5	8	8	15	12	11	6	10	15	0.319084	99/100	NonOverlappingTemplate
5	11	6	10	17	8	11	6	13	13	0.162606	100/100	NonOverlappingTemplate
11	15	7	6	12	5	9	17	8	10	0.145326	99/100	NonOverlappingTemplate
7	10	11	11	10	11	6	15	10	9	0.798139	99/100	NonOverlappingTemplate
9	10	9	10	20	8	5	11	6	12	0.085587	98/100	NonOverlappingTemplate
8	10	16	7	7	10	13	7	10	12	0.534146	99/100	NonOverlappingTemplate
7	8	8	16	15	11	8	7	14	6	0.191687	100/100	NonOverlappingTemplate
8	11	15	9	11	6	9	9	10	12	0.798139	97/100	NonOverlappingTemplate
11	5	7	12	11	7	11	12	10	14	0.637119	98/100	NonOverlappingTemplate
10	13	11	14	14	6	11	4	7	10	0.319084	98/100	NonOverlappingTemplate
11	9	13	7	6	14	10	11	11	8	0.759756	98/100	NonOverlappingTemplate
7	6	13	10	11	10	13	12	7	11	0.759756	100/100	NonOverlappingTemplate
9	9	7	7	16	8	14	11	8	11	0.514124	100/100	NonOverlappingTemplate
11	13	8	14	8	11	13	7	6	9	0.637119	97/100	OverlappingTemplate
11	12	8	10	10	9	12	8	8	12	0.978072	99/100	Universal
7	10	11	10	8	15	9	12	7	11	0.798139	98/100	ApproximateEntropy
2	2	5	1	7	3	3	4	10	5	0.057146	42/42	RandomExcursions
1	6	6	2	5	3	5	3	4	7	0.484646	41/42	RandomExcursions
6	5	4	5	4	4	3	6	3	2	0.911413	42/42	RandomExcursions
3	4	5	2	2	7	6	3	5	5	0.689019	42/42	RandomExcursions
4	4	8	5	1	3	8	6	1	2	0.090936	41/42	RandomExcursions
0	12	5	1	4	3	3	4	4	6	0.004301	42/42	RandomExcursions
2	4	7	3	6	5	3	5	3	4	0.788728	42/42	RandomExcursions
2	8	2	6	6	4	4	3	4	3	0.484646	42/42	RandomExcursions
3	0	4	7	5	2	5	2	10	4	0.035174	42/42	RandomExcursionsVariant
3	2	4	2	4	3	5	6	9	4	0.350485	41/42	RandomExcursionsVariant
3	2	2	2	4	4	4	5	4	12	0.021262	41/42	RandomExcursionsVariant
2	2	3	3	1	9	3	6	8	5	0.057146	42/42	RandomExcursionsVariant
1	3	6	4	6	2	3	5	3	9	0.186566	42/42	RandomExcursionsVariant
2	4	4	5	4	3	0	9	6	5	0.162606	41/42	RandomExcursionsVariant
4	1	3	6	4	7	3	6	4	4	0.637119	42/42	RandomExcursionsVariant
3	4	4	5	5	1	3	8	5	4	0.585209	41/42	RandomExcursionsVariant
4	4	6	4	2	6	4	4	1	7	0.585209	41/42	RandomExcursionsVariant
4	4	4	5	6	4	2	5	3	5	0.964295	42/42	RandomExcursionsVariant
3	2	7	3	3	7	2	6	7	2	0.242986	42/42	RandomExcursionsVariant
3	5	5	7	1	6	3	5	3	4	0.637119	42/42	RandomExcursionsVariant
3	6	5	5	7	5	3	4	1	3	0.637119	42/42	RandomExcursionsVariant
2	4	4	4	9	6	6	3	1	3	0.213309	41/42	RandomExcursionsVariant
1	3	3	6	6	3	8	3	5	4	0.392456	42/42	RandomExcursionsVariant
2	2	4	1	7	5	4	5	5	7	0.392456	42/42	RandomExcursionsVariant
2	3	2	3	5	2	4	7	6	8	0.275709	42/42	RandomExcursionsVariant
3	2	3	5	3	2	3	11	3	7	0.035174	42/42	RandomExcursionsVariant
10	11	7	12	8	5	8	12	16	11	0.455937	99/100	Serial
10	10	7	7	11	16	8	10	13	8	0.616305	99/100	Serial
8	11	9	7	14	10	10	10	10	11	0.955835	99/100	LinearComplexity

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 39 for a sample size = 42 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing A.9: Risultati del test NIST STS 2.1 su una sequenza di $100 \times 10^6\,\mathrm{bit}$ generati dal TRNG sulla mappa M_{ADC} a 4 rami, mostrata in Fig. 5.8. I 2 bit prodotti ad ogni iterazione della mappa vengono memorizzati a coppie. Un'operazione di XOR produce il valore binario di uscita. Lo schema di codifica impiegato è quello alternativo, 01-00-10-11-01.

Bibliography

[1] *1363-2000 - IEEE Standard Specifications for Public-Key Cryptography*. 2000.

[2] Lasota A and Mackey M.C. *Chaos, Fractals and Noise. Stochastic Aspects of Dynamics*. Heidelberg, Germany: Springer-Verlag, 1995. 2nd ed.

[3] Thomas Biege. Analysis of a strong pseudo random number generator by anatomizing linux' random number device. 2006.

[4] NY Bitforms Gallery, New York. Rand corporation - a million random digits with 100,000 normal deviates (1955), 2010. `http://www.bitforms.com/vanishing-point/rand-corporation-a-million-random-digits-with-100000-normal-deviates`.

[5] Digi-Key Corporation. Search digikey, 2014. `http://www.digikey.com/`.

[6] Rand Corporation. *A Million Random Digits with 100,000 Normal Deviates*. RAND Corporation, 2001.

[7] Lind D and Marcus B. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, New York, NY, USA, 1995.

[8] R. Devaney. *An introduction to Chaotic Dynamical Systems*. MA: Addison Wesely, 1989. 2nd ed.

[9] Simtec Electronics. Araneus alea i: True random number generator, 2009. `http://www.entropykey.co.uk/`.

[10] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley, 2003.

[11] Ben Fry and Casey Reas. Processing, 2009. http://processing.org/.

[12] Setti G, Mazzini G, Rovatti R, and Callegari S. Statistical modeling of discrete-time chaotic processes: Basic finite-dimensional tools and applications. In *Circuits and Systems, 2003. Tutorial Guide: ISCAS 2003. The IEEE International Symposium on*, volume 2, pages 68–92, May 2003.

[13] Ian Goldberg and David Wagner. Randomness and netscape browser, 1996. http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html.

[14] Solomon W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.

[15] Zvi Gutterman and Dahlia Malkhi. Hold your sessions: An attack on java session-id generation. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2005.

[16] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. Analysis of the linux random number generator. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 371–385, Washington, DC, USA, 2006. IEEE Computer Society.

[17] M. Hamburg, P. Kocher, and M. Marson. Analysis of intel's ivy bridge digital random number generator. *Cryptography Research*, 2012.

[18] IETF. Defending against sequence number attacks, 1996. https://www.ietf.org/rfc/rfc1948.txt.

[19] IETF. Rfc 4086 on randomness recommendations for security (replaces earlier rfc 1750), 2005. http://www.ietf.org/rfc/rfc4086.txt.

[20] Microchip Technology Inc. Single-ended, rail-to-rail i/o, low-gain pga, 2004. http://ww1.microchip.com/downloads/en/DeviceDoc/21908a.pdf.

[21] Microchip Technology Inc. Pic18f2455/2550/4455/4550 data sheet, 2009. http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf.

[22] Microchip Technology Inc. Pic18f2455/2550/4455/4550 family silicon errata and data sheet clarification, 2009. `http://ww1.microchip.com/downloads/en/DeviceDoc/80478a.pdf`.

[23] Microchip Technology Inc. Mcp4901/4911/4921 8/10/12-bit voltage output d/a converter w/ spi, 2010. `http://ww1.microchip.com/downloads/en/DeviceDoc/22248a.pdf`.

[24] Texas Instruments Incorporated. Lmv341, lmv342, lmv344 rail-to-rail output cmos operational amplifiers (rev. h), 2012. `http://www.ti.com/lit/gpn/lmv341`.

[25] Texas Instruments Incorporated. 17mhz, low noise, cmos in, 1.8v op amp w/ shutdown (rev. f), 2013. `http://www.ti.com/lit/gpn/lmv791`.

[26] Texas Instruments Incorporated. Lmv601/602/604 1mhz low pwr gen purpose, 2.7v oper amps (rev. b), 2013. `http://www.ti.com/lit/gpn/lmv601`.

[27] Maxim Integrated. Max600x - low-cost, low-power, low-dropout, sot23-3 voltage references, 2013. `http://datasheets.maximintegrated.com/en/ds/MAX6001-MAX6005.pdf`.

[28] Benjamin Jun and Paul Kocher. The intel random number generator. *Cryptography Research*, 1999.

[29] Ashton Kevin. That 'internet of things' thing, in the real world things matter more than ideas. *RFID Journal*, 2009.

[30] Benedikt Kristinsson. Ardrand: The arduino as a hardware random-number generator. Master's thesis, Reykjavik University, 2011.

[31] Chengxin Liu. *Jitter in Oscillators with 1/f Noise Sources and Application to True RNG for Cryptography*. PhD thesis, Worcester Polytechnic Institute, 2006.

[32] Yu Liu, Mingyi Zhu, and Hong Guo. High-speed truly random number generation via entropy amplification. Technical report, CREAM Group, State Key Laboratory of Advanced Optical Communication, 2010.

[33] Degaldo-Restituto M, Medeiro F, and Rodriguez-Vazquez A. Nonlinear switched-current cmos ic for random signal generation. *Electronics Letters*, 29(25):2190–2191, Dec 1993.

[34] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

[35] R.G. Mende, L.C. Noll, and S. Sisodiya. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system, 1998. `http://www.google.com/patents/US5732138`.

[36] Kennedy MP, Rovatti R, and Setti G. *Chaotic Electronics in Telecommunications*, chapter 12, pages 397–442. CRC Press International Series on Computational Intelligence Series. Taylor & Francis, 2000.

[37] John Von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards - Applied Math Series*, 12, 1951.

[38] NIST. Security requirements for cryptographic modules (fips pub 140-2), 1994. `http://web.archive.org/web/20070817151620/http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf`.

[39] NIST. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications (nist special publication 800-22rev1a), 2010. `http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf`.

[40] NIST. Recommendation for the entropy sources used for random bit generation (nist special publication 800-90b), 2012. `http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf`.

[41] Araneus Information Systems Oy. Entropy key: A small, cheap, and easy device that provides your computers and servers with extra performance, security, and reliability, 2009. `http://www.araneus.fi/products-alea-eng.html`.

[42] Benjamin Pousse. Short communication: An interpretation of the linux entropy estimator. *IACR Cryptology ePrint Archive*, 2012:487, 2012. informal publication.

[43] Bitcoin Project. Android security vulnerability, 2013. `https://bitcoin.org/en/alert/2013-08-11-android`.

[44] QuintessenceLabs. qstream: Quantum random number generator, 2013. `http://quintessencelabs.com/`.

[45] Cryptography Research. Evaluation of VIA C3 Nehemiah Random Number Generator. *Cryptography Research*, 2003.

[46] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems.

[47] Callegari S. Introducing complex oscillation based test: an application example targeting analog to digital converters. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 320–323, May 2008.

[48] Callegari S and Setti G. Adcs, chaos and trngs: a generalized view exploiting markov chain lumpability properties. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 213–216, May 2007.

[49] Callegari S, Rovatti R, and Setti G. Efficient chaos-based secret key generation method for secure communications. In *Proceedings of NOLTA*, October 2002.

[50] Callegari S, Rovatti R, and Setti G. Embeddable adc-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. *Signal Processing, IEEE Transactions on*, 53(2):793–805, Feb 2005.

[51] Callegari S, Rovatti R, and Setti G. First direct implementation of a true random source on programmable hardware. *International Journal of Circuit Theory and Applications*, 33(1):1–16, 2005.

[52] ID Quantique SA. Quantis: True random number generator exploiting quantum physics, 2010. `http://www.idquantique.com/random-number-generators/products.html`.

[53] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[54] Nick Sullivan. Ensuring randomness with linux's random number generator, 2013. `http://blog.cloudflare.com/ensuring-randomness-with-linuxs-random-number-generator`.

[55] Kuusela T. Random number generation using a chaotic circuit. *Journal of Nonlinear Science*, 3(1):445–458, 1993.

[56] Flying Stone Technology. Neug, a true random number generator implementation, 2012. `http://www.gniibe.org/memo/development/gnuk/rng/neug`.

[57] Folkert van Heusden and Damien Miller. Audio entropy daemon, 2009. `http://www.vanheusden.com/aed/`.

[58] W. A. Wagenaar. Generation of random sequences by human subjects: A critical survey of the literature. *Psychological Bulletin*, pages 65–72, 1972.